



UNIVERSIDAD  
DE MÁLAGA

Departamento de Arquitectura de Computadores

TESIS DOCTORAL

# High Performance Computing in the Cloud

Óscar Torreño Tirado


Julio de 2017

Dirigida por:  
Oswaldo Trelles



UNIVERSIDAD  
DE MÁLAGA

AUTOR: Óscar Torreño Tirado

 <http://orcid.org/0000-0001-8513-3109>

EDITA: Publicaciones y Divulgación Científica. Universidad de Málaga



Esta obra está bajo una licencia de Creative Commons Reconocimiento-NoComercial-SinObraDerivada 4.0 Internacional:

<http://creativecommons.org/licenses/by-nc-nd/4.0/legalcode>

Cualquier parte de esta obra se puede reproducir sin autorización pero con el reconocimiento y atribución de los autores.

No se puede hacer uso comercial de la obra y no se puede alterar, transformar o hacer obras derivadas.

Esta Tesis Doctoral está depositada en el Repositorio Institucional de la Universidad de Málaga (RIUMA): [riuma.uma.es](http://riuma.uma.es)

Dr. D. Oswaldo Trelles Salazar.  
Profesor Titular del Departamento de  
Arquitectura de Computadores de la  
Universidad de Málaga.

**CERTIFICA:**

Que la memoria titulada “High Performance Computing in the Cloud”, ha sido realizada por D. Óscar Torreño Tirado bajo mi dirección en el Departamento de Arquitectura de Computadores de la Universidad de Málaga y constituye la Tesis que presenta para optar al grado de Doctor en Ingeniería Informática.

Málaga, Julio de 2017



Dr. D. Oswaldo Trelles Salazar  
Director de la tesis.





A mis padres, pareja y familiares



# Agradecimientos

---

Esta tesis es el producto de la colaboración directa e indirecta de multitud de personas a las que quisiera dirigir unas breves palabras de agradecimiento con estas líneas.

En primer lugar, quiero dar las gracias a mi director, Oswaldo Trelles por acogerme en su grupo de investigación ya antes de terminar la licenciatura y por sus consejos y bien hacer en la dirección de la tesis, sin la cual el contenido de esta tesis hubiera sido de menor calidad. Me gustaría resaltar el clima de trabajo que ha predominado en el grupo de investigación, con un trato de igual a igual. He de dar las gracias también a Oswaldo por la dirección de los trabajos de fin de carrera y fin de máster.

Me gustaría extender este agradecimiento a todas las personas que integran el Departamento de Arquitectura de Computadores de la Universidad de Málaga, y en especial a los técnicos y a la secretaria, Carmen, cuyo trabajo y ayuda ha sido indispensable para que esto llegara a buen fin. Me han brindado un gran apoyo en momentos personales difíciles y me gustaría agradecerse de corazón.

Debo agradecer la financiación recibida por parte del Instituto de Salud Carlos III a cargo de los proyectos de investigación “Instituto Nacional de Bioinformática” (INB-GN5), RIRAAF (RD07/0064/0017 y RD12/0013/0006) y Plataforma tecnológica de Recursos Biomoleculares y Bioinformáticos (PT13/0001/0012), y a la Junta de Andalucía a través del proyecto Plataforma computacional de alto rendimiento para la gestión y análisis de datos clínico-genéticos (P10-TIC-6108).

I must thank the European Commission funded project Mr.Symbiomath, under the 7th framework programme grant agreement no. 324554, whose funds made me enjoy a two year investigation stay at RISC Software GmbH company part of the Johannes Kepler University, with the group of my host, Mag. Michael T. Krieger, formed by Bashar Ahmad, Paul Heinzlreiter and Iris Leitner. Thank you for your kind support.

I would also like to thank the whole company and the secretaries for their invaluable help and support. Without the cloud resources provided by RISC it would have been much harder to develop the work described in this thesis.

Por último, quisiera agradecer encarecidamente el soporte anímico y afectivo que me ha brindado toda la gente de mi entorno. A mis antiguos compañeros de grupo Maxi García, Alfredo Martínez, Victoria Martín, Javier Ríos y Antonio Muñoz, sin su ayuda no habría sido posible la fácil adaptación al grupo. A mis compañeros actuales del grupo Jose Antonio Arjona, Sergio Díaz, Esteban Pérez y Eugenia Ulzurrun con los cuales he tenido buenas discusiones de trabajo. Al resto de compañeros de laboratorio que han creado un ambiente de trabajo inigualable, Ricardo Quislan, Alberto Sanz, Antonio J. Dios, Miguel Ángel González, Manolo R. Cervilla, Manolo Pedrero, Antonio Vilches, Alejandro Villegas, Jose Manuel Herruzo y a todos aquellos que haya podido olvidar. A mis amigos de Ronda, Málaga y Austria, que han supuesto un gran apoyo en momentos difíciles y una ineludible válvula de escape para el estrés. A Cristina, que ha sido un apoyo incondicional y ha estado a mi lado en todo momento, aguantando todos mis altibajos. Y a mis padres y familiares, especialmente a mi madre porque aunque no esté ya con nosotros se sentirá orgullosa de mí allá donde esté.

# Abstract

---

The numerous technological advances in data acquisition techniques allow the massive production of enormous amounts of data in diverse fields such as astronomy, health and social networks. Nowadays, only a small part of this data can be analysed because of the lack of computational resources. High Performance Computing (HPC) strategies represent the single choice to analyse such overwhelming amount of data. However, in general, HPC techniques require the use of big and expensive computing and storage infrastructures, usually not affordable or available for most users.

Cloud computing, where users pay for the resources they need and when they actually need them, appears as an interesting alternative. Besides the savings in hardware infrastructure, cloud computing offers further advantages such as the removal of installation, administration and supplying requirements. In addition, it enables users to use better hardware than the one they can usually afford, scale the resources depending on their needs, and a greater fault-tolerance, amongst others. The efficient utilisation of HPC resources becomes a fundamental task, particularly in cloud computing. We need to consider the cost of using HPC resources, specially in the case of cloud-based infrastructures, where users have to pay for storing, transferring and analysing data. Therefore, it is really important the usage of generic tasks scheduling and auto-scaling techniques to efficiently exploit the computational resources. It is equally important to make these tasks user-friendly through the development of tools/applications (software clients), which act as interface between the user and the infrastructure.

In recent years, the interest in both scientific and business workflows has increased. A workflow is composed of a series of tools, which should be executed in a predefined order to perform an analysis. Traditionally, these workflows were executed in a manual way, sending the output of one tool to the next one in the analysis process. Many applications to execute workflows automatically, appeared recently. These applications ease the work of the users while executing

their analyses. In addition, from the computational point of view, some workflows require a significant amount of resources. Consequently, workflow execution moved from single workstations to distributed environments such as Grids or Clouds. Data management and tasks scheduling are required to execute workflows in an efficient way in such environments.

In this thesis, we propose a cloud-based HPC environment, focusing on tasks scheduling, resources auto-scaling, data management and simplifying the access to the resources with software clients. First, the *cloud computing infrastructure* is devised, which includes the base software (i.e. OpenStack) plus several additional modules aimed at improving authentication (i.e. LDAP) and data management (i.e. GridFTP, Globus Online and CloudFuse). Second, built on top of the mentioned infrastructure, the *TORQUE distributed resources manager* and the *Maui scheduler* have been configured to schedule and distribute tasks to the cloud-based workers. To reduce the number of idle nodes and the incurred cost of the active cloud resources, we also propose a configurable *auto-scaling technique*, which is able to scale the execution cluster depending on the workload. Additionally, in order to simplify tasks submission to the TORQUE execution cluster, we have interconnected the *Galaxy workflows management system* with it, therefore users benefit from a simple way to execute their tasks. Finally, we conducted an *experimental evaluation*, composed by a number of different studies with synthetic and real-world applications, to show the behaviour of the auto-scaled execution cluster managed by TORQUE and Maui.

All experiments have been performed by using an OpenStack cloud computing environment and the benchmarked applications correspond to the benchmarking suite, which is specially designed for workflows scheduling in the cloud computing environment. Cybershake, Ligo and Montage have been the selected synthetic applications from the benchmarking suite. GECKO and a GWAS pipeline represent the real-world test use cases, both having a diverse and heterogeneous set of tasks.

# Contents

---

<b>Agradecimientos</b>	<b>I</b>
<b>Abstract</b>	<b>III</b>
<b>Contents</b>	<b>v</b>
<b>List of Figures</b>	<b>XI</b>
<b>List of Tables</b>	<b>XV</b>
<b>List of Abbreviations</b>	<b>XVII</b>
<b>1.- Introduction</b>	<b>1</b>
1.1. Motivation and Contributions . . . . .	1
1.2. Research objectives . . . . .	3
1.3. Contributions . . . . .	3
1.4. Scope and limitations . . . . .	5
1.5. Outline of the Thesis . . . . .	6
<b>2.- Background and Related Work</b>	<b>7</b>
2.1. Cloud computing . . . . .	7



2.1.1.	Cloud computing service models . . . . .	9
	Infrastructure-as-a-Service . . . . .	11
	Platform-as-a-Service . . . . .	11
	Software-as-a-Service . . . . .	11
2.1.2.	Cloud computing deployment models . . . . .	11
	Public cloud . . . . .	12
	Private cloud . . . . .	12
	Community cloud . . . . .	13
	Hybrid cloud . . . . .	13
2.2.	Scheduling . . . . .	13
2.2.1.	Overview of tasks scheduling . . . . .	13
2.2.2.	Scheduling algorithms . . . . .	14
	Multilevel queue scheduling . . . . .	15
	Priority-based scheduling . . . . .	15
2.2.3.	Distributed resource managers . . . . .	15
2.3.	Workflows . . . . .	18
2.3.1.	Workflow management systems and software clients . . . . .	19
	Workflow management systems . . . . .	19
	Software clients . . . . .	22
2.4.	Application Domain . . . . .	23
2.4.1.	Pairwise sequence comparison . . . . .	23
2.4.2.	Genome-wide association studies . . . . .	24
2.5.	Auto-scaling strategies . . . . .	25
2.6.	Workflows scheduling in the cloud . . . . .	34
<b>3.-</b>	<b>Infrastructure</b>	<b>37</b>
3.1.	Cloud computing solution: OpenStack . . . . .	37
3.2.	Authentication . . . . .	39



3.3. Data management . . . . .	42
3.3.1. Underlying file system (Ceph) . . . . .	42
3.3.2. Globus Online . . . . .	43
3.3.3. Endpoint Setup . . . . .	44
GridFTP User authentication . . . . .	44
Object Storage Integration . . . . .	45
Endpoint Registration . . . . .	46
Architecture and Workflow . . . . .	47
3.4. Computation . . . . .	47
3.4.1. TORQUE distributed resources manager . . . . .	48
3.4.2. RESTful Web Services front-end . . . . .	49
3.4.3. Galaxy workflows management system . . . . .	51
Galaxy deployment . . . . .	54
3.5. Networking . . . . .	55
3.6. OpenStack Horizon . . . . .	58
3.7. Interconnection between components . . . . .	58
<b>4.- Scheduling and auto-scaling</b>	<b>63</b>
4.1. Scheduling . . . . .	63
4.1.1. Built-in TORQUE FCFS scheduler . . . . .	64
4.1.2. Maui scheduler . . . . .	64
Job priority parameters . . . . .	64
Node allocation policy . . . . .	65
Backfill . . . . .	67
4.2. Auto-scaling strategy . . . . .	70
4.2.1. Configuration parameters . . . . .	71
4.2.2. Scaling decision mechanism . . . . .	72
4.2.3. The deployment scenario . . . . .	74

<b>5.- Experimental Evaluation</b>	<b>75</b>
5.1. Workflow applications . . . . .	75
5.1.1. Synthetic workflows . . . . .	75
5.1.2. Real-world workflows . . . . .	79
Pairwise genome comparison workflow . . . . .	80
Multiple genome comparison parallel workflow . . . . .	81
Genome-Wide Association Study workflow . . . . .	83
5.2. Evaluation metrics . . . . .	83
5.3. Experiment results . . . . .	84
5.3.1. Results of the performance metrics . . . . .	85
Queued time . . . . .	85
Makespan . . . . .	87
Throughput . . . . .	89
Resource utilisation . . . . .	89
5.3.2. System behaviour using the auto-scaling strategy . . . . .	90
Multiple genome comparison sequential workflow . . . . .	91
Multiple genome comparison parallel workflow . . . . .	91
Genome-Wide Association Study workflow . . . . .	96
5.4. Main factors affecting the scheduling and auto-scaling mechanisms	96
5.4.1. Task duration distribution . . . . .	97
5.4.2. Inaccuracies in the task runtime estimates . . . . .	99
5.4.3. Provisioning delays . . . . .	105
<b>6.- Conclusions and Future Work</b>	<b>109</b>
<b>Appendices</b>	<b>117</b>
<b>A.- Cloud computing features</b>	<b>117</b>

A.1. On-demand . . . . . 117

A.2. Pay-per-use . . . . . 117

A.3. Rapid elasticity . . . . . 118

A.4. Maintenance and upgrading . . . . . 119

**B.- Scheduling 121**

B.1. Types of processes . . . . . 121

B.2. Scheduling level . . . . . 122

    B.2.1. Short-term . . . . . 122

    B.2.2. Medium-term . . . . . 122

    B.2.3. Long-term . . . . . 123

B.3. Types of scheduling algorithms (static and dynamic) . . . . . 124

B.4. Traditional scheduling algorithms . . . . . 126

    B.4.1. FIFO . . . . . 126

    B.4.2. Shortest Job First . . . . . 127

    B.4.3. Round-robin scheduling . . . . . 127

**C.- Resumen en español 129**

C.1. Introducción . . . . . 129

C.2. Estado actual de desarrollo . . . . . 132

    C.2.1. Computación en la nube . . . . . 132

    C.2.2. Planificación de tareas . . . . . 133

    C.2.3. Flujos de trabajo . . . . . 134

    C.2.4. Dominios de aplicación . . . . . 135

    C.2.5. Trabajos relacionados . . . . . 135

C.3. Infraestructura . . . . . 136

    C.3.1. OpenStack . . . . . 136

    C.3.2. Autenticación . . . . . 136

    C.3.3. Gestión de datos . . . . . 137



C.3.4. C3mputo . . . . . 138

C.3.5. Infraestructura de red . . . . . 139

C.3.6. Horizon . . . . . 139

C.3.7. Interconexi3n entre componentes . . . . . 139

C.4. Planificaci3n y auto-escalado . . . . . 140

    C.4.1. Planificaci3n de tareas . . . . . 140

    C.4.2. Auto-escalado . . . . . 141

C.5. Evaluaci3n experimental . . . . . 141

    C.5.1. Factores que afectan la planificaci3n y el auto-escalado . . . 142

C.6. Conclusiones y trabajo futuro . . . . . 143

**Bibliography** **147**



# List of Figures

---

2.1. Cloud computing service models. . . . .	10
2.2. Comparison of what users and what cloud providers control in the different models. . . . .	10
2.3. Workflow example. . . . .	18
2.4. Architecture of the Amazon Web Services cloud auto-scaling strat- egy. . . . .	26
2.5. Architecture of the IBM cloud auto-scaling strategy. . . . .	27
2.6. Architecture of the OpenStack cloud auto-scaling strategy. . . . .	27
2.7. Architecture of the OpenNebula cloud auto-scaling strategy. . . . .	28
2.8. Architecture of the Microsoft Azure cloud auto-scaling strategy. . . . .	28
2.9. Current architecture of the “Distributed Infrastructure with Re- mote Agent Control” (DIRAC) distributed infrastructure. . . . .	30
2.10. Architecture of the Cloud Scheduler auto-scaling strategy. . . . .	31
2.11. Architecture of the Cloud Scheduler auto-scaling strategy. . . . .	32
2.12. Architecture of the Dynamic Terascale Open-source Resource and QUEUE Manager (TORQUE) auto-scaling strategy working in ac- tive mode. . . . .	33
2.13. Architecture of the Dynamic TORQUE auto-scaling strategy work- ing in passive mode. . . . .	33
3.1. OpenStack components. . . . .	39
3.2. Ceph Object Gateway. . . . .	43

3.3. Interaction of all components to mount the users corresponding containers of the object storage. . . . .	46
3.4. Architecture of the Endpoint Setup including the object storage integration. . . . .	47
3.5. RESTful Web Services front-end. . . . .	50
3.6. Invoking a REpresentational State Transfer (REST)ful Web Service. . . . .	51
3.7. GECKO workflow in Galaxy. . . . .	52
3.8. Parameters of GECKO workflow in Galaxy. . . . .	53
3.9. GECKO workflow running in Galaxy. . . . .	54
3.10. Physical architecture of the networking component of the devised cloud environment. . . . .	56
3.11. Virtual architecture of the networking component of the devised cloud environment. . . . .	57
3.12. Overview page in Horizon . . . . .	58
3.13. Overview of the system architecture. . . . .	60
4.1. Example scenario of job delays caused by backfill. . . . .	68
4.2. Best-fit backfill algorithm. . . . .	70
4.3. Overview of Dynamic TORQUE running in active mode. . . . .	71
4.4. Deployment of Dynamic TORQUE in the described cloud infrastructure. . . . .	74
5.1. Overview of the Ligo workflow. . . . .	76
5.2. Overview of the Montage workflow. . . . .	77
5.3. Overview of the Cybershake workflow. . . . .	78
5.4. The pairwise genome comparison workflow. . . . .	80
5.5. The parallelization levels applied to the multiple genome comparison workflow. . . . .	82
5.6. The GWAS workflow. . . . .	83
5.7. Average queued time of the different workflows scheduled by the two compared scheduling algorithms. . . . .	86

5.8. Makespan of the different workflows scheduled by the two compared scheduling algorithms. . . . .	88
5.9. Performance of the scheduling algorithms for an unbalanced set of long tasks. . . . .	88
5.10. Throughput of the different workflows scheduled by the two compared scheduling algorithms. . . . .	89
5.11. Resources utilisation of the different workflows scheduled by the two compared scheduling algorithms. . . . .	90
5.12. Speedup of the first parallelization level of the multiple genome comparison workflow. . . . .	93
5.13. Speedup of the modules composing the GECKO workflow. . . . .	95
5.14. Histogram of tasks duration for the different workflows. . . . .	98
5.15. Barchart of tasks duration for the different workflows. . . . .	99
5.16. Histogram of tasks duration for the different workflows including an extra random time. . . . .	101
5.17. Barchart of tasks duration for the different workflows including a random extra time. . . . .	102
5.18. Average queued time of the different workflows scheduled by the two compared scheduling algorithms including a random extra time.	103
5.19. Makespan of the different workflows scheduled by the two compared scheduling algorithms including a random extra time. . . . .	104
5.20. Influence of the inaccuracies in the tasks runtime estimates to the makespan metric. . . . .	105
5.21. Provisioning delay of the OpenStack instance types. . . . .	107
B.1. Illustration of in what part of a computing system a short-term, medium-term or long-term scheduler acts. . . . .	124
B.2. Taxonomical classification of task scheduling algorithms [17]. . . . .	125





# List of Tables

---

2.1. Limitations of related workflows scheduling algorithms. . . . . 36

5.1. Dataset used to evaluate the performance of GECKO. . . . . 81



# List of Abbreviations

---

- AMQP** Advanced Message Queuing Protocol. 48
- API** Application Programming Interface. 5, 26–28, 43, 48, 49, 74, 111, 116
- CA** Certification Authority. 44, 47, 137
- CAPEX** Capital Expenses. 9, 117
- CAR** Computación de Alto Rendimiento. 129–131, 133, 143–145
- CLI** Command Line Interface. 26, 27
- CPU** Central Processing Unit. 5, 14, 25, 28, 29, 34, 36, 48, 66, 96, 113, 121, 122, 127, 135, 136
- CSS** Custom Style Sheets. 54
- DBaaS** Database-as-a-Service. 59, 61
- DHCP** Dynamic Host Configuration Protocol. 57
- DIRAC** “Distributed Infrastructure with Remote Agent Control”. 29, 30
- DNAT** Destination Network Address Translation. 56
- DNS** Domain Name System. 61
- E/S** Entrada/Salida. 135
- EaaS** Everything-as-a-Service. 9
- EC2** Elastic Compute Cloud. 29, 49, 106

- FCFS** First-Come First-Served. 4, 39, 63, 64, 79, 84–86, 88–90, 97, 102, 126
- FIFO** First-In First-Out. 126, 140, 142, 143
- FIM** Federated Identity Management. 40
- FP7** Seventh Framework Programme. 116
- FPGA** Field Programmable Gate Array. 122
- GO** Globus Online. 43, 44, 137, 138
- GPU** Graphical Processing Unit. 122
- GRE** Generic Routing Encapsulation. 55, 57
- GWAS** Genome-wide association studies. 24, 135
- HPC** High Performance Computing. 1, 2, 15–17, 23, 25, 38, 59, 65, 109, 110, 113–116, 123
- HPSs** High-scoring Segment Pairs. 80
- HTC** High Throughput Computing. 48
- HTTP** Hypertext Transfer Protocol. 43, 60
- I/O** Input/Output. 5, 36, 76, 78, 82, 93, 94, 109, 113, 121, 122
- IaaS** Infrastructure-as-a-Service. 3–5, 9, 11, 35, 37, 38, 41, 49, 59, 109, 110, 115, 133, 136
- ISI** Institute for Scientific Information. 5, 132
- JCR** Journal Citation Reports. 5, 132
- KVM** Kernel-based Virtual Machine. 48, 138
- LDAP** Lightweight Directory Access Protocol. 3, 37, 40, 41, 45, 110, 137, 144
- MAPI** Modular API. 22, 23, 50, 51
- MIT** Massachusetts Institute of Technology. 31
- ML2** Modular Layer 2. 55

- MOM** Machine Oriented Mini-server. 32, 74
- NFS** Network File System. 60, 61, 74, 141
- OPEX** Operational Expenses. 9, 118
- PaaS** Platform-as-a-Service. 9, 11, 38, 109, 133
- PAM** Pluggable Authentication Module. 45, 47
- PBS** Portable Batch System. 32, 48, 74
- POSIX** Portable Operating System Intefarce. 44
- QoS** Quality of Service. 8
- RAM** Random Access Memory. 34, 48, 96
- REST** REpresentational State Transfer. 4, 37, 39, 49, 51, 113, 138
- S3** Simple Storage Service. 43
- SA** scheduling algorithm. 13, 14
- SaaS** Software-as-a-Service. 9, 11, 109, 133
- SDKs** software development kits. 11
- SDN** Software-Defined Networking. 55
- SFTP** SSH File Transfer Protocol. 113
- SGE** Sun Grid Engine. 31
- SJF** Shortest Job First. 15
- SLA** Service Level Agreement. 8, 12, 118, 119
- SLURM** Simple Linux Utility for Resource Management. 49
- SNAT** Source Network Address Translation. 56
- SNPs** Single Nucleotide Polymorphisms. 24, 83
- SSH** Secure Shell. 41

**SSL** Secure Sockets Layer. 41

**TCP** Transmission Control Protocol. 43

**TORQUE** Terascale Open-source Resource and QUEue Manager. 4–6, 18, 32, 33, 37, 41, 48, 49, 61, 63, 64, 66, 70–72, 74, 84, 100, 110–112, 114, 115, 138–141, 144

**URL** Uniform Resource Locator. 4, 50

**VCF** Variant Call Format. 83, 96

**VM** Virtual Machine. 4, 5, 28–32, 34, 36, 41, 49, 55, 57, 60, 61, 71, 72, 74, 96, 97, 106, 112, 114, 115

**VOs** Virtual Organisations. 40, 41

**WMSs** Workflow Management Systems. 19–22

**xinetd** Extended Internet Daemon. 45

# 1 Introduction

---

This chapter provides a high-level overview of the work presented in this thesis. It starts off by providing the motivation behind data intensive applications and workflow management systems in the context of cloud computing (Section 1.1). Then it provides the research objectives set at the beginning of this thesis summarising and explaining their achievement (Section 1.2). The chapter also describes the contributions of the presented work to the scientific community (Section 1.3). It finally concludes with the scope of the presented work, its limitations, how they could be enhanced in future work (Section 1.4), and provides an outline of the thesis chapters (Section 1.5).

## 1.1. Motivation and Contributions

The large number of technological advances in data acquisition techniques allow the massive production of enormous amounts of data in diverse research fields such as astronomy, health and social networks. This data offers unprecedented opportunities in research to research groups and companies. However, at present only a small part of this data –the top of the iceberg– can be synthesized, managed and processed, providing just a partial understanding of the involved processes [62, 87]. The poor availability of computational resources and approaches to efficiently exploit them are the major bottlenecks in results acquisition.

An interesting and increasingly important alternative to face the current pace of data acquisition is the use of High Performance Computing (HPC) strategies due to their long story of success dealing with large datasets and computational demanding applications. These strategies cover data management techniques,

how to access the data, as well as the scheduling and distribution of tasks analysing the previously mentioned data. In general, HPC techniques require the use of big and expensive computing and storage infrastructures, usually not affordable for most of the research groups requiring these resources [67].

Cloud computing appears as an interesting alternative where users pay for the resources they need and when they actually need them. Besides the savings in infrastructure, cloud computing offers further advantages such as the removal of installation, administration and supplying requirements. In addition, it allows to use better hardware than the one users can usually afford buying, the flexibility for resources scaling depending on user needs and a greater fault-tolerance, among others.

Given the enormous amount of data being generated and the associated computational load to manage and process it, the efficient utilisation of HPC resources becomes a fundamental task, not easy at all if we want to obtain the highest possible performance. Without forgetting the incurred costs for storing and transferring data, in cloud computing is especially important the efficient use of the resources we have asked for, because this will later determine the response time until final results acquisition and the monetary costs to be covered [75]. Therefore, it is really important to use generic tasks scheduling techniques, which should allow to extract a better efficiency from the computational resources.

Additionally, one of the greatest drawbacks to adopt these new technologies is their difficult installation, configuration and use [8]. Therefore, it is equally important to make these tasks easier through the development of tools/applications (software clients), which allow the use of the low-level developed tools, acting as interface between the parties (i.e. the end-user and the actual infrastructure).

Comparative genomics, related to biomedicine, does not escape from the trends of massive data production and the high-demand of processing power and associated costs. Many comparative genomics applications work with input and/or output data in the range of gigabytes, terabytes and exceptionally petabytes.

Besides, the tasks arisen from comparative genomics analyses are complex, heterogeneous, and with data dependencies between them, usually executed in what is known as workflows. These characteristics make them appropriated to be used as a validation set for data transfer and storage techniques, and also for tasks scheduling in the proposed cloud-based HPC computing environment.

The research in solutions to the challenge of processing big datasets coming from biomedicine and bioinformatics in a cloud computing environment, is of



direct interest in life sciences, medicine and in particular, health care. From the technological point of view, this work addresses challenges in data transfer and storage, as well as tasks scheduling and easing the use of cloud computing resources to end-users.

## 1.2. Research objectives

The research objectives of the performed work are (in chronological order):

- Infrastructure selection: types of cloud computing offers with their advantages and disadvantages. Evaluation of the most significant cloud computing providers.
- Identification of the tasks to be performed in the cloud computing environment (e.g. data transfer, browsing of available services/tools, navigation through the hierarchy of available services, execution of tasks, etc.).
- Strategies for tasks scheduling, distribution and load-balancing.
- Software clients: design/implementation of data uploading/downloading software for the cloud environment. Moreover, software to execute tasks in the cloud (including monitoring, results retrieval, etc.).
- Use case in the comparative genomics field (multiple genome comparison) and in the biomedical domain (genotype calling).
- Benchmarking the performance and quality of the devised strategies.

## 1.3. Contributions

The main contributions of this work are the following:

- The use of an Infrastructure-as-a-Service (IaaS) *cloud computing infrastructure*, which includes the base software (i.e. OpenStack) plus several additional modules aimed at improving authentication (i.e. Lightweight Directory Access Protocol (LDAP)) and data management (i.e. GridFTP, GlobusOnline and CloudFuse).

- The configuration, installation and deployment of a *TORQUE*<sup>1</sup> and *Maui*<sup>2</sup>-managed execution cluster built on top of the previously mentioned cloud computing infrastructure. The configuration and start-up of the execution cluster has been fully automated using OpenStack instance snapshots and Heat templates to define start-up installation scripts.
- The design and implementation of an *auto-scaling technique* independent from the used IaaS cloud computing solution, although it has been only tested with the mentioned OpenStack cloud infrastructure. The implemented mechanism has been tested with the TORQUE distributed resources manager, with its default First-Come First-Served (FCFS) scheduler and with the Maui scheduler. The auto-scaling technique offers many configuration possibilities such as the minimum number of static worker nodes and the maximum number of dynamic ones. Other configurable parameters are the polling interval to check for idle nodes and the number of jobs required to be waiting in the queue for running in order to instantiate a new Virtual Machine (VM).
- The design and development of a *generic RESTful Web Services front-end* to allow the execution of tools in a cloud computing environment. The simple REST interface allows job submission (returning a job id), progress monitoring and intermediate and final results retrieval. Independently from the given tool, the interface shares the endpoints of the mentioned operations, changing only the base Uniform Resource Locator (URL), which points to the specific tool.
- The interconnection of the mentioned auto-scaled execution cluster with existing *software clients* (i.e. jORCA, mORCA) and *workflow management system* (i.e. Galaxy) in order to simplify browsing the available services/workflows, and interconnecting and executing them via graphical user interfaces.
- The development of two pairwise and multiple genome comparison applications (i.e. GECKO and GECKO in parallel), which are not only two good use cases to evaluate the system. In addition, they represent an important contribution to the bioinformatics domain since they remove the limitation on input sequence length and execution time faced by equivalent software. The results of GECKO are of comparable or higher quality compared to the output produced by similar tools (as demonstrated in the publication).

---

<sup>1</sup><http://www.adaptivecomputing.com/products/open-source/torque/>

<sup>2</sup><http://www.adaptivecomputing.com/products/open-source/maui/>

- An *experimental evaluation* is carried out to compare our proposals with the available solutions in the presence of tasks of different nature (Central Processing Unit (CPU)-bounded and Input/Output (I/O)-bounded, and with regular and irregular computational patterns) and with different cloud environment configurations (i.e. instance types). A *thorough analysis and simulation* of the developed auto-scaling technique is also performed, including the configuration parameters of the TORQUE distributed resources manager using benchmarking applications used widespread in the literature, and additionally comparative genomics and biomedicine workflows.

The aforementioned contributions have been published in international peer-reviewed conferences [40, 81, 85] and journals [84, 52] ranked by the Institute for Scientific Information (ISI) Journal Citation Reports (JCR) as required in the rules of the PhD programme. Additional contributions have helped in the achievement of this thesis and in the training of writing scientific papers [82, 83].

## 1.4. Scope and limitations

The main contributions of this work have been listed in the previous section, however we would like to further clarify the scope and main limitations. First, as cloud computing infrastructure, the devised work uses OpenStack. Although theoretically prepared to be used with any other IaaS cloud solution, it has not been tested in such infrastructures. In principle, when the cloud management Application Programming Interface (API) is not fully compatible with the OpenStack API, a simple adaptation of the cloud management API component would be required. Second, the chosen distributed resources manager has been TORQUE alongside the Maui job scheduler. Again, these tools can be replaced requiring in the first place the configuration of VM instances with the new tools installed, and in second place, new Heat deployment templates for the automatic instantiation of the nodes. Third, the configuration parameters of the scheduler have been selected in order to improve the system behaviour within the devised cloud computing infrastructure and for the synthetic and real-world workflows presented in Chapter 5. There is not a clear limitation in this point, the set of tasks used to test the system has been big and heterogeneous enough, and actually any kind of application can be used in the system. However, further tests with more real-world workflows would reinforce the performance of the system. In summary, all the developed components could be easily replaced, given the designed architecture, where all the components have been interconnected through clearly defined interfaces.

## 1.5. Outline of the Thesis

The remainder of the thesis is structured in 5 additional chapters as follows:

- *Chapter 2* provides the necessary background to understand the starting point of the performed work in addition to the related work alternatives present in the state of the art. More concretely, the chapter provides background on cloud computing, scheduling, workflows, workflow management systems and distributed resource managers.
- *Chapter 3* describes the used cloud computing infrastructure including information about data management, authentication and computation. The infrastructure and the components comprising it were selected taking into account previous studies on the available cloud solutions (both public and private). This chapter concludes with the explanation of how the different parts of the system are interconnected.
- *Chapter 4* presents the scheduler to be used in conjunction with the TORQUE distributed resource manager and its corresponding parameters. Additionally, the cloud computing resources auto-scaling technique is described.
- *Chapter 5* presents first the selected evaluation metrics to benchmark the scheduling algorithm and the auto-scaling technique. Secondly, the workflows, representing both simulated and real workloads, are described. Thirdly and finalising this chapter, the results obtained for the different evaluation metrics in the various workflows are presented and discussed.
- *Chapter 6* outlines the conclusions extracted from this work and sets possible new future research directions.

# 2 Background and Related Work

---

This chapter presents a background on cloud computing starting from several definitions of the term and mentioning its unique features and the different available service and deployment models in Section 2.1. Appendix A describes in more detail the unique features of cloud computing listed in this chapter. Section 2.2 provides the background on tasks scheduling related to the scheduling mechanism designed in this thesis. Appendix B includes a more detailed background of the categorisation of the traditional scheduling algorithms. Next, Section 2.3 include an overview of workflows, workflow management systems and software clients and their usage in the cloud computing environment. A brief background of the application domains for the real-world workflows developed and used in this thesis is contained in Section 2.4. A more detailed background of the application domains of such tools can be consulted in [88, 86]. Finally, this chapter presents the related work on auto-scaling techniques and workflows scheduling in the cloud in Sections 4.2 and 2.6.

## 2.1. Cloud computing

Nowadays, even some years after the appearance of cloud computing, there is not a clear common definition about what cloud computing is. There is no consensus between end-users and cloud providers what exactly this term means. Taking a look at the existing definitions of cloud computing, we may find common parts of what this term involves or might involve. Here we quote three definitions for cloud computing:

- (1) Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing

resources (e.g. networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. – U.S. National Institute of Standards and Technology [64].

**(2)** A computing Cloud is a set of network enabled services, providing scalable, Quality of Service (QoS) guaranteed, normally personalised, inexpensive computing infrastructures on demand, which could be accessed in a simple and pervasive way. – Lizhe WANG *et al.* [95].

**(3)** Clouds are a large pool of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services). These resources can be dynamically reconfigured to adjust to a variable load (scale), allowing also for an optimum resource utilisation. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the Infrastructure Provider by means of customised Service Level Agreement (SLA)s. – Luis M. Vaquero *et al.* [91].

As it can be extracted from the previous definitions, the term cloud computing comprise different types of services. These services include storage and processing power available as a service over the network. Although there are differences in the previously written definitions, it is possible to extract several common features. These features are described in more detail in Appendix A.

Finally, before moving to explain its main features, it is worth noting that cloud computing does not consist on an entirely new technology, instead is a mixture of two main already available solutions: virtualization and grid computing. The former provides virtual simulated versions of computing, storage and network resources, among others. The latter, grid computing, uses several computers interconnected over the network to solve a given problem. Grid computing has its origins in the nineties when Foster and Kesselman introduced it and later formalise it in 1999 [30]. Typically it is used to solve scientific problems requiring a big number of processing units and/or involving large amounts of data. Grid systems are architected in a way that a single user can request a large part of the whole infrastructure. In contrast, the main focus of public cloud computing providers is the provision of small portions of the infrastructure to end-users, maximising the concurrent number of users. However, cloud computing in itself does not limit users to request large parts of the infrastructure if they can afford it.

*On-demand* is one of the basic features defining cloud computing. In cloud computing, computing and storage resources can be started/terminated whenever the users need them. This capability removes the necessity of planning ahead the amount of required resources, thus reducing the cost of unused resources. An additional important feature of the cloud environment is its billing model. It translates the Capital Expenses (CAPEX) of buying the required infrastructure, into Operational Expenses (OPEX) measured as the time cloud resources have been used. Despite having several types of resources to meet user requirements, these requirements frequently vary along time. Cloud providers offer automatic techniques to either up-/down-scale the resources depending on the workload, what is commonly referred as *elasticity*. They maintain the underlying physical computing resources, this turns into an effective outsourcing of maintenance tasks (either hardware or software). For instances, in case of hardware failures or scheduled maintenance tasks, virtual machines are migrated to different computing resources, thus not affecting customers' experience.

### 2.1.1. Cloud computing service models

Cloud computing has offered three different service models since its appearance. These models are usually denominated in the service-oriented architecture domain as Everything-as-a-Service (EaaS). The taxonomy is organised in a stack (see Figure 2.1), from a higher abstraction level to a lower one, being “E” Software, Platform or Infrastructure [64]. It is worth noting that the different models are not completely independent, instead Software-as-a-Service (SaaS) is built on Platform-as-a-Service (PaaS), and so does the latter on IaaS. Each of the mentioned levels are explained in detail in next subsections. Although these are the main models present in the cloud computing environment, some authors differentiate even further the models, for example talking about Workflows-as-a-Service [94], Database-as-a-Service or Hardware-as-a-Service [74]. Figure 2.2 summarises the differences.

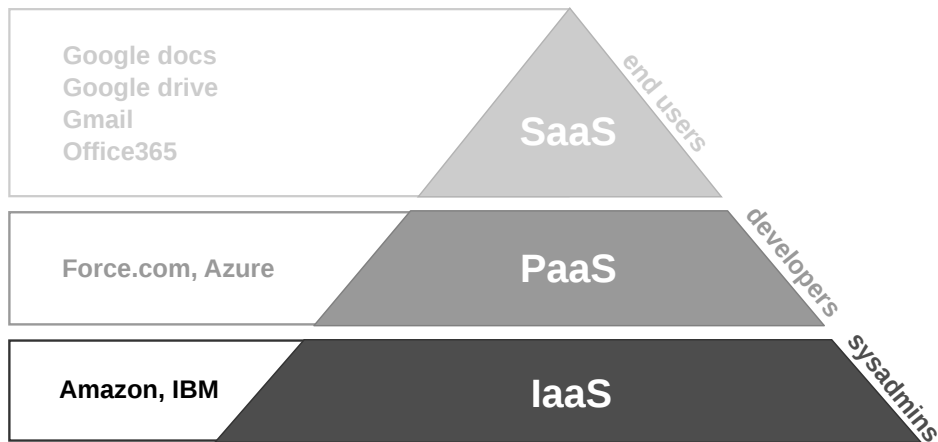


Figure 2.1: Cloud computing service models.

IaaS	PaaS	SaaS	
Applications	Applications	Applications	User managed
Runtimes	Runtimes	Runtimes	
Security	Security	Security	
Databases	Databases	Databases	
Servers	Servers	Servers	Vendor managed
Virtualization	Virtualization	Virtualization	
Storage	Storage	Storage	
Networking	Networking	Networking	

Figure 2.2: Comparison of what users and what cloud providers control in the different models.



### **Infrastructure-as-a-Service**

IaaS is the delivery of a computing infrastructure as a service. In this cloud model users receive raw computing, storage and network resources where they can run any operating system or application of their choice. In simple words, IaaS provides a higher flexibility from the point of view of the user (compared to PaaS and SaaS). This higher flexibility resides mainly in the software the user can install after instantiating a machine. Despite the higher flexibility, users are not able to decide a specific hardware platform or change parameters of the underlying infrastructure. They can only choose which of the data centers or availability zones of the cloud provider they would like to use.

### **Platform-as-a-Service**

In this model, cloud providers go beyond providing the hardware and virtual machines running on top of them. They are providing software development kits (SDKs) with support for different programming languages. The goal of this model is to provide a higher abstraction level to the users while building their applications. PaaS is typically used by developers looking for an easy way to implement their software, which does not require special configurations to be run.

### **Software-as-a-Service**

In this last model, cloud providers offer either one or a set of final applications running on-demand on a cloud infrastructure. The target customers of this model are mainly end-users, who are usually only interested in using the software and not in what is behind the scenes to support it. The applications following this model are typically accessed through a simple client interface, such as a Web browser. This is the simplest model from the point of view of the end-user. However, in this case configuration capabilities are strongly limited to those of the served application.

#### **2.1.2. Cloud computing deployment models**

Cloud environments can also be classified depending on the underlying infrastructure deployment model as Public, Private, Community or Hybrid. Each of these deployment models can be distinguished by several factors. A first factor

is the targeted users (whole community, company workers, etc.). Another factors defining the deployment model are the architecture and location of the physical data center where the cloud is constructed. And last but not least are the needs of the customers (usually related to legal concerns).

## Public cloud

A *public cloud* consists on a publicly available infrastructure owned by a cloud service provider. In this deployment model there are no restrictions in the users able to use it, neither on the applications they can run on it. In this model, users pay for their resource utilisation in a pay-per-use way as explained in Section A.2 of Appendix A.

## Private cloud

A *private cloud* is built for the exclusive use of one customer (either a single individual or a company). The customer owns and has full control of the cloud environment, what turns into the distinguishing characteristic of a private cloud. However, there are different private cloud environments depending on how they are operated.

Although a private cloud is usually owned by a single individual or a company, it can be built, installed and managed by a third party company. Similarly, the physical servers hosting the environment can be located at the facilities of the customer or elsewhere within a shared hosting facility.

Building a private cloud is obviously more expensive in terms of equipment, installation and maintenance compared to using a public cloud. The difference in cost is even higher if you are buying, hosting and maintaining the physical servers. As a consequence, customers requiring the use of a private cloud, sometimes ask cloud providers to supply them a private cloud environment hosted within the same facilities of the public cloud servers. Undoubtedly the SLAs and legal concerns of this kind of private cloud would be different to the one of the regular public cloud offerings. The main reasons are that in private clouds resources are used by far less people compared to public clouds, thus the SLAs play a less important role. In the case of legal concerns, private cloud users know that their data does not leave the security borders defined by their company or organisation, thus they do not need special legal contracts in this matter.

### Community cloud

Because of the higher cost of private clouds, and also as a result of several customers having similar requirements, *community clouds* appeared. In this model, customers share the infrastructure with the consequent reduction in the operational costs (i.e. installation, configuration and management). Therefore, there is not a single owner of the infrastructure but a community (i.e. a group of companies/customers) owning it (e.g. Salesforce community cloud<sup>1</sup>).

### Hybrid cloud

A *hybrid cloud* is made out of any composition of the previously mentioned deployment models. The most typical scenario is the mixture of the private and public deployment models. In this scenario users store and manage their sensible data within the private part of the environment, whereas less sensible data is stored and processed in the public part. This is typically done to alleviate the use of the possibly smaller private part. Although this is the most typical hybrid scenario, any other combination is also considered a hybrid cloud.

## 2.2. Scheduling

### 2.2.1. Overview of tasks scheduling

Nowadays, scheduling happens in many facets of our lives. Scheduling is needed because physical resources (e.g. time, computers, etc.) are limited, and typically people want to extract the best possible performance out of their resources. In computer science, scheduling refers to the method of assigning work to be done to computing resources. Depending what we need to schedule and at what level, the work may be threads, processes, tasks or workflows (defined as a group of tasks with a given set of dependencies, thus executed in a predefined order).

A scheduling algorithm (SA) performs the scheduling activity. The goal of each SA usually differs. For example, some SAs pay attention to the load balancing, acting as a fair judge distributing the work between all the computing resources. While other SAs concentrate on providing a fair usage of the resources to users belonging to a multi-user system. Scheduling is a fundamental piece of a

---

<sup>1</sup><https://www.salesforce.com/editions-pricing/community-cloud/>

computing system. It happens at many levels ranging from the operating system within a regular computer, to distributed computing environments.

SAs typically aim at one or several goals. Frequent goals are for example, maximising throughput (amount of work finished per time unit), reducing makespan or latency (the elapsed time between a task entering the system for execution and its subsequent completion), minimising the response time (defined as the elapsed time from submission to the start of the execution), and maximising fairness (appropriate CPU time for each process/user in concordance with the user defined priority values). In practice, these objectives often conflict, therefore schedulers find a compromise between them based on user preference.

In some situations, SAs must ensure that processes meet user-defined deadlines. A typical scenario where this is of vital importance is in real-time environments. In such scenario, if processes do not comply with the defined deadlines there could be unacceptable consequences. Not meeting deadlines may risk human lives in embedded systems in automotive or airplanes, or waste a lot of money in embedded systems designed to control production lines in industry. Similarly in a scientific environment, not complying to deadlines may cause a waste of resources while executing a set of tasks. In particular, in multiple steps analyses the results produced by intermediate tasks might become useless if the deadline is surpassed.

In a cloud computing environment a new variable to be optimised enters the game. This variable is the incurred cost of executing a set of tasks. It is not that in former systems there is no cost, but the cost in such systems is the indirect one of not making an efficient use of the resources. For example, in grid computing there was a lot of research optimising tasks scheduling to maximise the throughput [103, 14]. However, nowadays in public cloud environments there is a direct cost when using resources. As a consequence, new scheduling algorithms for the cloud environment [90, 59] aim to minimise the cost of executing a set of tasks.

### 2.2.2. Scheduling algorithms

Tasks scheduling has been a research field with a significant amount of work since the first years of computing. Bad scheduling decisions might compromise the efficiency of single-core systems and particularly the behaviour of distributed systems. As a consequence, a number of schedulers have been developed in the field. Next subsections describe the concept of the scheduling algorithm applied in this thesis. Common and basic knowledge about tasks scheduling has

been included as an Appendix (see Appendix B). This appendix describes the traditional scheduling algorithms developed over the years for Operating Systems, on which most of the current state-of-the-art algorithms are inspired.

### **Multilevel queue scheduling**

This kind of schedulers is used in systems where processes can be easily divided into groups. Traditionally, the most common division has been interactive and batch processes, but nowadays this division has evolved to more than two levels/queues in certain environments. In any case, the idea is to separate processes in different queues paying attention to the specific features of each type of process. By doing this, it is possible to implement different methods for each of the queues trying to optimise the system behaviour.

### **Priority-based scheduling**

Although the Shortest Job First (SJF) scheduler is somehow basing its decisions in a priority value calculated as the shortest execution time, there exist a series of algorithms with different ways of calculating the job priority. For example, there are algorithms calculating the job priorities based on the estimated execution time of the tasks. Others calculate the tasks priorities based on other factors such as the time the job has been queued in the system, the amount of dependencies they resolve upon termination, etc.

Regardless of the factors the scheduler uses to calculate the priority, priority-based schedulers can be categorised in dynamic and static priority algorithms. The difference resides in if the job priority is updated as the system evolves (dynamic) or if it does not change since the job enters the system (static).

Independently of the factors used to calculate the job priority and also of its dynamic/static value, most priority-based algorithms work in a similar way. This kind of algorithms keeps the ready queue sorted by job priority, dispatching each scheduling cycle the job with highest priority. Similarly to the SJF algorithm, in case of a preemptive scheme, low-priority jobs (analogous to long jobs in SJF) might enter into starvation if higher priority jobs enter continuously the system.

#### **2.2.3. Distributed resource managers**

In distributed computing environments such as HPC clusters, grids and clouds the presence of an entity to manage the set of available resources is required. The

price reduction in commodity off-the-shell hardware and the good scalability of the cluster architecture made it feasible to build large scale clusters with thousands of processors.

An essential component that is needed to manage such large-scale distributed computing infrastructures is a resource management system. The main aim of resource management in distributed environments is to make sure that users can use such systems as easy as they can access local resources. However, the functionality required by a distributed resource manager is wider in scope. Typically, the main competences of such a system are:

- **Checking nodes' health and status:** in distributed systems it is particularly important for both administrators and end-users to know the status of the different nodes composing the system. Administrators will need to fix failing nodes, whereas end-users will know the number of available nodes working properly.
- **Task scheduling:** HPC environments are typically very expensive to build and maintain, therefore, they are simultaneously used by different people. As in any other shared system, an independent and fair entity is required to control the use of the resources. The main aims of task scheduling algorithms in distributed computing environments are: minimisation of the average job waiting time in the queue, and throughput and resources utilisation maximisation. Other optimisation objectives could be targeted, but the previous three are the most common. Distributed resource managers typically allow configuring different schedulers, but the most used is the priority-based scheduler.
- **Matching tasks to physical resources:** once a given task has been selected by the tasks scheduler to be executed, the next step is deciding on which node(s) it will be executed. The decision at this point will later determine the system behaviour. For instance, if we have a distributed system with 2 nodes, the first with 4 cores and the second with 8, if both are free and a job asking for 4 nodes enters the system, it is better to assign it to the node with 4 cores. This action allows for a later assignment of jobs asking for more than 4 cores to be executed in the second node. This is just an artificial and dummy example, but it illustrates that wrong resources allocation would increase the waiting time of upcoming jobs and compromise efficient resources utilisation.

The previous points represent the main competences of a distributed resource management system. An ideal resource manager should be provided following a

series of design goals:

- **Simplicity:** the distributed resource management system should have a sufficient and simple set of features to allow users to work in a distributed environment similarly to how they work on their local computers.
- **Portability:** it should be written in a general purpose programming language, which can be deployed in most operating systems and hardware architectures. This facilitates porting of applications between distributed systems controlled by the same manager.
- **Scalability:** a distributed resource manager should be constructed to allow handling clusters with up to thousands of nodes. Similarly, the number of simultaneous jobs it is able to handle should be equally high. Nowadays, big HPC clusters (e.g. TOP500 supercomputers<sup>2</sup>) are composed of thousands of nodes, allowing many users to work simultaneously. Therefore, the number of jobs in such systems is very high, thus requiring an efficient and scalable management.
- **Configurability:** it should provide as many configuration possibilities as possible. For instance, what schedulers can be used, or how the scheduler calculates the job priority, how jobs are assigned to physical hosts, etc. This is important to allow users configuring the system to their specific needs in order to extract the best possible performance.
- **Extensibility:** ideally the resource manager should provide a way to incorporate new functionality without the need of modifying the base source code (e.g. via plug-ins). For example, not supported architectures could be easily included by developing a plug-in.
- **Robustness:** hardware or software failures are somewhat frequent, particularly in systems composed of a big number of nodes. Resource managers should handle different failures by resuming the work from the last valid point (i.e. a checkpoint) or at least informing the user that his execution has failed. Typical failures handled by resource managers are node problems (if a node becomes unresponsive) and file system issues.
- **Security:** distributed resource managers should provide a secure authentication mechanism to authenticate users within the system. Additionally, they should control which nodes can communicate and interact with each other. Especially in distributed systems, foreign malicious nodes could compromise the security of the whole system.

---

<sup>2</sup><https://www.top500.org/>

- System administrator friendly: it should be configured with as few configuration files as possible, and ideally from a single node. Distributed configuration files would make the configuration process more difficult. It should be possible to change the system configuration without affecting the running jobs. A complete set of tools should be provided to facilitate the system monitoring and administration.

In this thesis, we have selected the open source distributed resource manager TORQUE, which implements the three mentioned competences of a resource management system, following the design goals contained above.

## 2.3. Workflows

The complexity of nowadays analyses in various research disciplines motivate the interconnection of several data analysis steps to analyse the data following a divide-and-conquer manner. Typically there exists a sufficient number of tools to analyse the input data, it is just required to execute them in the correct order to produce the desired output. The orchestrated execution of several analysis tools is commonly referred as workflows. Figure 2.3 contains a simple workflow example.

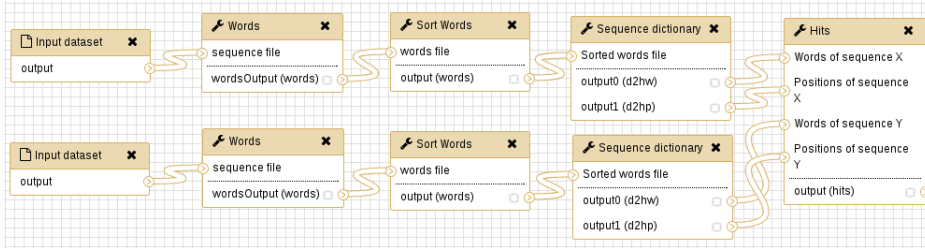


Figure 2.3: Workflow example containing two independent branches and a final common step. Each of the boxes represent a different tool (i.e. Words, Sort Words, Sequence Dictionary and Hits). The pipes connecting the tools symbolise data dependencies between the tools.

In recent years, interest in workflows has raised, both in the business and scientific domains. The way workflow management systems handle workflows appear as a nice way to automatically execute a set of applications relieving the users of the technical impediments that such task could arise, such as the transfer



and translation of data from one tool to another. Whilst this can be performed manually, it is more practical to automate this process.

Some tools part of workflows require a large amount of computational power, which is usually not available within a single workstation. Consequently, the execution of workflows was moved to distributed execution environments such as grids or clouds. For an efficient usage of such environments, the execution should be conveniently planned and scheduled. For this purpose, scheduling techniques have been studied in the grid computing environment over many years [103, 22], a trend that can be also observed nowadays in the cloud computing environment [27, 58].

Another important problem to be addressed is to ensure that experts on a number of different fields, in some cases with limited knowledge in computing, can process their data in a reproducible and portable manner via user-friendly solutions.

### 2.3.1. Workflow management systems and software clients

#### Workflow management systems

Workflow Management Systems (WMSs) appeared as a way to execute workflows easier in a reproducible and portable way, not only in regular workstations but also in distributed computing environments. WMSs have experienced a year-over-year increase in their popularity and availability. The main objective of WMSs is allowing the composition and deployment of workflows, which piece together separate analysis steps often carried out by different software packages. However, the functionality implemented in current WMSs is wider.

WMSs' tasks include workflow definition, execution and management driven by the representation of the workflow logic [53]. These systems facilitate creating the representation of the workflow logic, entering the input data of the analysis process, monitoring its execution and results retrieval. When the user enacts a workflow, WMSs invoke the tools forming the workflow in the predefined order, managing the data dependencies between these. The software component responsible for enacting the workflow is called a Workflow Enactor [36].

The main reason of using WMSs is the advantages they provide to the researcher. In first place, WMSs remove the necessity of manually executing a multi-step analysis. Additionally, features such as fault tolerance, data provenance recording, tools meta-data, visual creation of workflows, results exploration, storage of intermediate results, user profiles and sharing of resources com-

plete the advantages of using WMSs. Further details of the mentioned advantages are described below.

- Automated execution: the removal of manual operations, such as copying, editing and pasting data between tasks greatly speeds up the time taken to perform an analysis [28]. In addition, this frees up time for the researcher to carry out other analyses in parallel.
- Automated task-level parallelisation: many WMSs such as Swift [100] provides automatic task-level parallelisation in order to automatically execute in parallel tasks without dependencies in the underlying infrastructure.
- Fault tolerance: software and hardware failures are unfortunately a common occurrence. Some WMSs help mitigating such failures by automatically restarting the execution of the failed step or resuming the workflow execution from the last valid analysis step.
- Data provenance recording: WMSs typically keep a comprehensive record of every task performed, including the used software, the version, input data and the selected parameters [7]. This allows the user to keep track of the different performed analyses, and enhances the reproducible property of the analysis that have been carried out.
- Tools meta-data: workflows can be complex especially when they are composed of many services. In addition, the analysis services could be rather sophisticated requiring extensive parameterisation in their use. WMSs are able to store and later retrieve used values of the mentioned parameters as well as other meta-data.
- Visual creation of workflows: most recent WMSs allow the construction of workflows through a graphical canvas with drag and drop tools, thereby removing the need to manually write the workflow definition, which is usually stored in markup languages such as XML. This feature greatly reduces the required users' expertise to construct the analysis pipeline of their interest.
- Results exploration: once the workflow execution has come to an end, WMSs allow users retrieving and exploring the results. Both, text and graphical representations of the most commonly known formats are typically included in the majority of the WMSs to enable further post-analysis.
- Storage of intermediate results: data generated by the internal analysis steps is stored. This allows first to explore and determine if the internal steps are being correctly executed, and second, to perform thorough or

parameter-sensitive analyses by slightly changing the execution parameters of the tool(s) of interest.

- **User profiles:** some WMSs are provided as traditional desktop applications where there is only one user profile typically stored in the user hard disk. However, modern web-based WMSs implement user accounting systems, providing users a secure environment in which to store and analyse their data.
- **Reproducible research:** achieving reproducible research is one of the aspects most of the funding agencies, journals and scientific entities aim for. Data provenance recording alone is not sufficient, resources (data and analysis tools) sharing is an indispensable feature if we want to achieve reproducible research. One way or another, traditionally workflows and data have been shared but requiring the use of multiple tools. Current WMSs allow to share workflows and data with the user community allowing a given analysis to be easily repeated. Additionally, sharing workflows and data with the user community within a single tool undoubtedly reduces the required time to perform an analysis, because users do not need to learn two different tools and neither they have to move information from one tool to the other.

It should be noted that whilst it is undeniably clear that WMSs offer a great number of benefits to the researcher, they come at a price and there are some disadvantages associated with their use. These are the following:

- **Lack of portability:** although most WMSs allow sharing data and workflows between researchers, this is not the case for researchers using different WMSs. As different research groups might potentially use different WMSs based on their preference, this presents an issue as adapting workflows to run on different environments can be a complex and tedious process. Additionally, if a research group decides migrating to a new system, the process of adapting existing workflows to the new system becomes a problem.
- **Public WMSs oversubscribed:** the popularity of publicly available WMSs over the net translates them into not viable solutions for many reasons. Long waiting times for workflows to be executed, quotas for the number of jobs and the amount of data that can be stored, bottlenecks and limitations uploading and downloading data especially under a slow internet connection, toolset not containing the required tools for the dataset of interest, and in general depending on a third party to have the system always available for the users' needs.

- Data privacy concerns: some research fields such as biomedicine work with sensitive data, which in some situations can not leave the context of a given research group or organisation, otherwise they would be violating national or international laws. Public WMSs typically implement security and data privacy mechanisms, but there is always the concern of what is happening with the data after leaving my organisation. Most public WMSs offer the possibility of installing their system in a trusted server, thus alleviating such concern.
- Lack of pre-existing components or wrapped tools: whilst a number of workflows and tools are typically available for various WMSs, it might occur that a particular analysis may need a tool, which is not available. This would require including the missing tool by writing some code, for which detailed knowledge of the WMSs underlying architecture is required. The previous would be only possible in the case that the WMSs allows including new tools, but not all of them have such functionality.

There exist several freely available WMSs, whilst there are also commercial ones. Although commercial WMSs may offer a viable paid alternative such as KNIME [11], and Pipeline Pilot [96], in this thesis we focus in the freely accessible, multidisciplinary and popular WMSs Galaxy [1].

## Software clients

Many research fields rely on the universal availability of web resources to analyse the data. These web resources represent important sources of information (databases) and numerous Web Services to perform different tasks. Web Services became really popular, but there was a need to facilitate services discovery, invocation, input standardisation and output visualisation. As a result, several clients to meet such requirements were developed, but nowadays with the appearance of new protocols and new required functionality there is a need to extend them.

jORCA [61] is one of the mentioned clients, designed to be used with bioinformatics and biomedicine Web Services. The application was specifically designed to help users take advantage of computational resources made available as Web Services, i.e. discover them, display available parameters, request information and finally execute the service. To enable all these activities, jORCA uses meta-data repositories (i.e. containers of meta-information), with information about available services and data types. The unification of meta-data information is provided by the Modular API (MAPI) library [71, 48]. By using this library, it is

possible to extend the execution functionality of jORCA with components called workers.

Within this thesis, we have enabled jORCA (and other MAPI-based clients such as MOWServ [72] and mORCA [25]) invoking Web Services deployed in cloud computing environments. These services can be provided by different institutes and their meta-data is stored in catalogues of services.

## 2.4. Application Domain

In this section a brief introduction to two particular applications of the bioinformatics and biomedicine research fields is provided. Only basic information to allow understanding the thesis is included, a more detailed description of the two particular applications explained in the two following subsections can be consulted in [88, 86]. Additionally, the mentioned material contain other related background, which might be of interest for the reader to better understand the use cases.

### 2.4.1. Pairwise sequence comparison

Pairwise sequence comparison is used to reveal functional, structural or evolutionary similarities between sequences. In the literature there exist several algorithms to perform the pairwise alignment. Such algorithms face limitations in the length of the input sequences they can deal with, because they report memory consumption and computational space issues. Even for linear algorithms, the analysis of long sequences such as the human genome makes the problem unapproachable in regular workstations, turning HPC environments in the only viable solution to address the alignment. However, from the biological point of view, their comparison models were designed for genes and proteins, thus they do not aim at finding translocations or inversions in the sequences under comparison.

The dot matrix is a simple way to understand pairwise sequence comparison. It consists in a two-dimensional array where the query sequence is arranged in one dimension and the reference sequence in the other. A dot is plotted in a given matrix entry if the residues of both sequences in that position are the same. Users can easily identify similar regions by locating that contiguous dots along the same diagonal.

The previous approach is valid for short sequences, but when the sequence length increased more specific algorithms were required to have a better overview

of the similarity shared by the input sequences. As a consequence, first the Needleman & Wunsch [65] global alignment, and second the Smith & Waterman [78] local alignment were developed. Both algorithms implement dynamic programming approaches with quadratic memory consumption and execution time.

On one hand, the global alignment is aimed at calculating an alignment, which involves all the residues present at the input sequences. In case the length of the sequences differ, the necessary gaps to continue the alignment are included. The alignment score typically depends on the number of inserted gaps, and on the sum of matches/mismatches using a value obtained from a scoring matrix. On the other hand, the local alignment determines similar regions of the input sequences, not necessarily being the entirety of the input sequences. Typically several local alignments are reported in the comparison of two sequences.

The previous algorithms allowed obtaining similarity measures of the input sequences, however, their complexity and the increasing length of the sequenced species required new implementations. As a result, a new family of algorithms appeared, not only targeting pairwise sequence comparisons, but also database searches given a query sequence and a database composed of a set of sequences. These algorithms (e.g. FASTA [56, 68], BLAST [5], BLAT [50]) calculate in a first step a set of word matches. These word matches or seeds are later used as starting points to calculate the alignment. The word length parameter determines both the algorithm sensitivity, and the execution time.

The mentioned tools represent an heterogeneous set of tasks with different computation patterns. Typically such applications have I/O intensive parts, such as loading the ever growing input data size, whereas they also contain pure CPU parts such as the calculation of the sequence alignment and its quality measures. This irregular set of tasks represents a good real-world use case to test scheduling mechanisms for distributed systems.

### 2.4.2. Genome-wide association studies

Genome-wide association studies (GWAS) are a relatively new way for scientist to identify genomic variations involved in human disease. Such studies aim to find small variations, called Single Nucleotide Polymorphisms (SNPs), that occur more frequently in individuals with a particular disease compared to people without the disease. Researchers analyse the results of the mentioned studies to determine genes that might contribute to a person's risk of developing a certain disease. GWAS examine SNPs along the genome, looking for single or groups of

variations that may contribute to a person developing a disease.

The typical size of the input data (i.e. a whole genome sequencing file) could be of 100GB per individual. The task to extract the genotypes or genetic variations out of these files can take around a week and will use approximately 1TB of temporary hard disk size per individual. As this task is completely independent for each individual, it represents an embarrassingly parallel problem and as such very suitable for a data-parallel execution. These tools often require an amount of computational resources greater than what is typically available in a moderate size hospital, which is a typical environment where such an analysis takes place. This combined with the varying amount of resources depending on the number of individuals to analyse make these tools particularly well suited to HPC environments. In addition, such analyses are usually performed by medical doctors or people from their research laboratories, therefore it is really important to make as easy as possible their execution, since they typically have a limited knowledge of the command-line, scripting, and programming languages.

## 2.5. Auto-scaling strategies

As the popularity of cloud computing has grown so have the efforts to make an efficient use of the provided resources. There exist several active developments, some provided in a native way by the public cloud vendors, in the private cloud solutions, and others non-native solutions developed to target specific cloud environments and distributed resource managers.

The native auto-scaling techniques provided by the public cloud providers and also in the private cloud solutions are somewhat similar. The similarity is particularly significant amongst the auto-scaling strategies provided in the Amazon, IBM, OpenStack and OpenNebula cloud solutions as can be observed in the Figures 2.4, 2.5, 2.6, 2.7. In the mentioned solutions, a set of instances are grouped together and treated as a logical unit. In these solutions alarms can be set up to be triggered on CPU and memory usage, and on specific points in time. These alarms are defined to decide under what situations the group should be scaled (either up or down). In the case of Microsoft Azure the alarm can only be set to the CPU usage or the number of messages present in specific internal communication queues as illustrated in Figure 2.8.

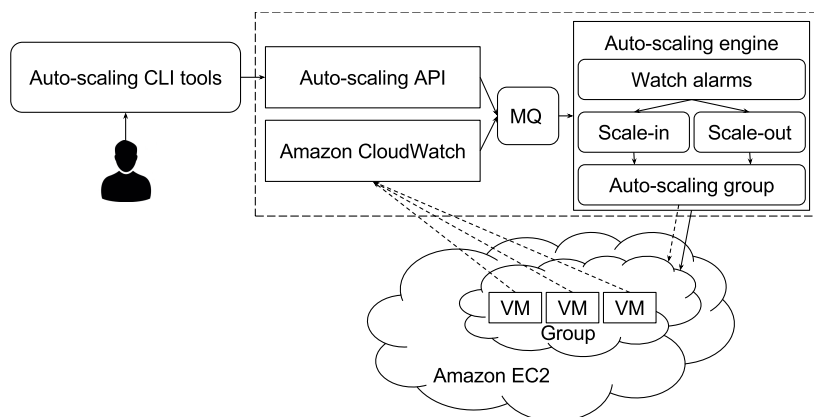


Figure 2.4: Architecture of the Amazon Web Services cloud auto-scaling strategy. ‘Auto-scaling API’ provides programmatic access to the auto-scaling functionality to the Command Line Interface (CLI) and Web tools. ‘Amazon CloudWatch’ monitors the specified metrics for all the instances in the auto-scaling group. The ‘Watch alarms’ define when the scale-in or the scale-out policy should be contacted. After the policy receives the action, ‘Auto-scaling group’ performs the scaling activity.



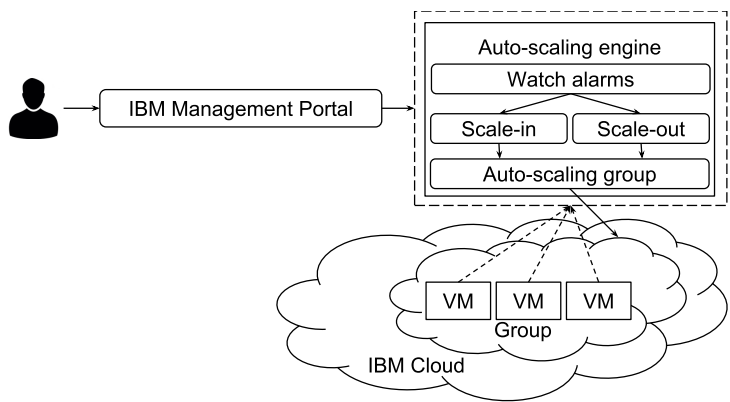


Figure 2.5: Architecture of the IBM cloud auto-scaling strategy. The auto-scaling policy could be defined via the ‘IBM Management Portal’ or using a provided API. Usage-based and schedule-based triggers could be defined to scale the system. Depending on these triggers or alarms, the scale-in or scale-out policy is invoked. One policy or the other, uses the ‘Auto-scaling Group’ component to perform the scaling.

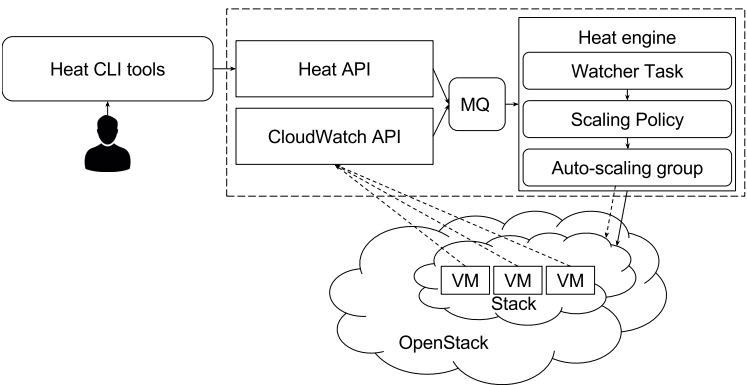


Figure 2.6: Architecture of the OpenStack cloud auto-scaling strategy. ‘Heat API’ provides programmatic access to the auto-scaling functionality to the CLI and Web (Horizon Dashboard) tools. ‘CloudWatch API’ receives samples from all the instances in the auto-scaling group or ‘Stack’. The ‘Watcher Task’ contacts the scale-in or the scale-out policy depending on the triggered alarm. After the policy receives the action, ‘Auto-scaling group’ performs the scaling activity.

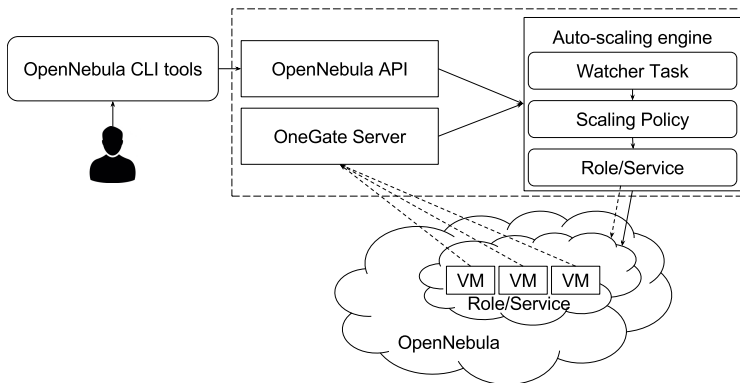


Figure 2.7: Architecture of the OpenNebula cloud auto-scaling strategy. The ‘OpenNebula API’ centralises the access to the OpenNebula auto-scaling mechanism. The ‘OneGate Server’ collects VM information such as CPU, memory and network usage. This information flows to the ‘Watcher task’, which scales the system depending on the configured alarms. The system can be scaled in three ways: change in the number of resources; setting a given cardinality; and percentage changes in the number of resources. Additionally schedule-based triggers could be configured.

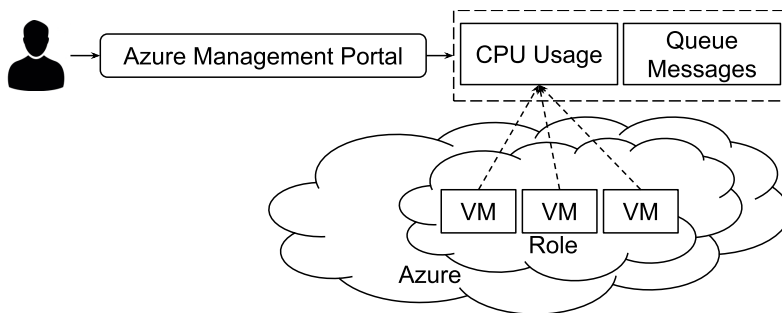


Figure 2.8: Architecture of the Microsoft Azure cloud auto-scaling strategy. The management of the auto-scaling policies is done through the Azure portal. In such portal, usage-based (CPU and length of message queues) or schedule-based alarms could be defined. VMs report their CPU usage and the aggregated value (average of the last 45 minutes) is compared with the user-defined alarms. If such value is above or below the thresholds, then the resources are scaled accordingly.

These native auto-scaling strategies are usually enough for some given services

but they are not sufficient for a cloud-based scientific computing environment. They have limitations with regards to the watching points to trigger alarms, and even there are solutions charging extra money apart from the cost of the cloud instances (such as IBM). However, the most significant limitation for a scientific computing environment is that the existing auto-scaling techniques are not aware of the number of processes neither of the resources the jobs are asking for. For instance, in an execution cluster with medium CPU usage, if a new CPU-bounded application enters the system, the performance of the rest of the applications could seriously degrade. In addition, if a new job requiring more cores than the available ones enters the system, the cluster will not be up-scaled, thus in the best case the application would execute slower and in the worst case its execution will just fail because of not having the required number of resources. Because of these limitations, a number of custom auto-scaling strategies have been developed. The most relevant ones related with the work presented in this thesis are explained in the following paragraphs.

The grid-middleware project: DIRAC [26] extended its pilot software agents to submit jobs to Amazon Elastic Compute Cloud (EC2)-compatible clouds. Upon VMs start, agents (pre-installed in the VMs) contact the central server requesting payload jobs. In order to control the used resources, DIRAC developed 3 additional services: A VM scheduler which monitors the task queues and instantiate new VMs when needed; a VM monitor, placed on each VM, which reports activity and shuts down the VM when no longer needed; and a VM manager, which preserves information about the VMs being used. It also provides usage monitoring through an extended version of the DIRAC web interface. This solution was used to control a collection of VMs with a total number of 800 cores located in the Amazon EC2 cloud [6]. The most recent architecture of the infrastructure is shown in Figure 2.9.

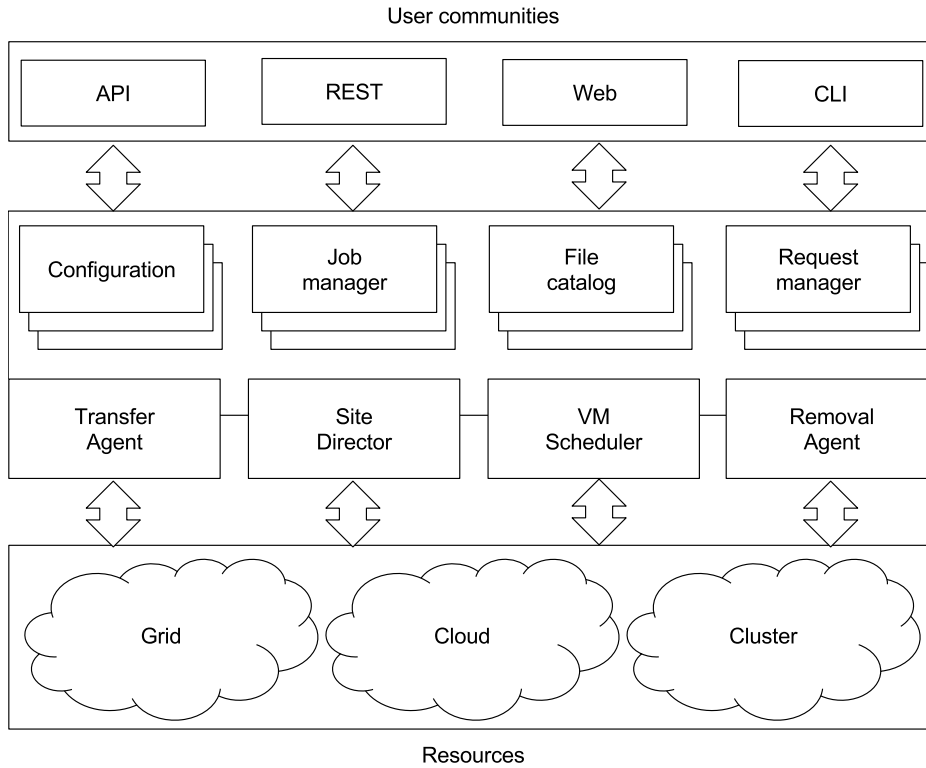


Figure 2.9: Current architecture of the DIRAC distributed infrastructure.

Another solution is Cloud Scheduler [9], which consists of a suite of Python scripts running alongside the Condor [80] resource manager. Similarly to the previous solution, Cloud Scheduler monitors the job queue and (if required) instantiates new VMs adding them to the Condor worker nodes pool (up to a certain limit). It also monitors the worker nodes, shutting them down as soon as the assigned jobs are done and no more jobs are suitable to be executed on them. This solution has been tested in the NeCTAR project of the Australian federal government to analyse data collected at the Large Hadron Collider at CERN [18]. Figure 2.10 illustrates the architecture of Cloud Scheduler.

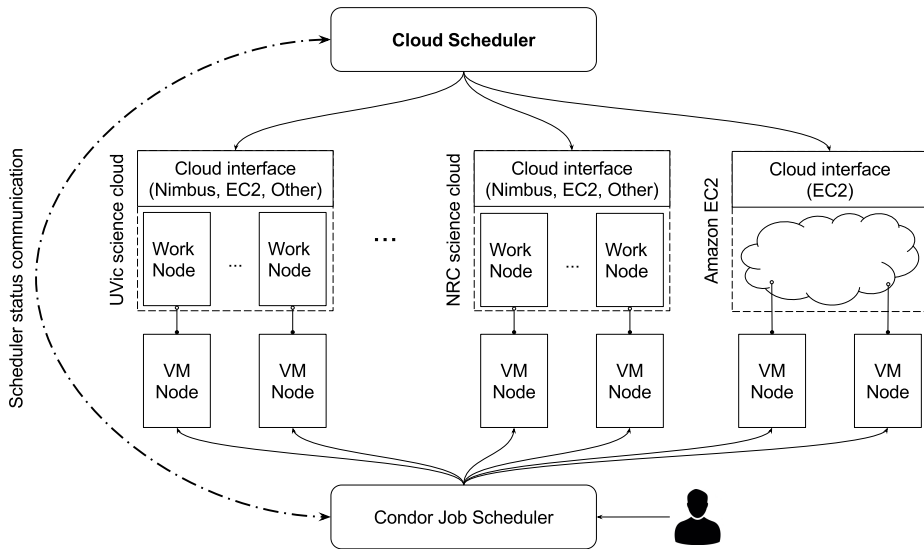


Figure 2.10: Architecture of the Cloud Scheduler auto-scaling strategy.

StarCluster [63] is an open source cluster manager developed by the Massachusetts Institute of Technology (MIT) for the Amazon cloud (see Figure 2.11). Its aim is the automation and simplification of managing clusters of VMs. Currently, it has support for a multiple number of applications, including the Sun Grid Engine (SGE) [35] and Condor distributed resource managers, and also many other software such as MySQL and Hadoop [99]. It configures a set of VMs grouped together as a cluster composed by one head node and a specific number of worker nodes. The head node exposes a shared file system which in turn can be accessed by the worker nodes. Once the resource manager is installed in all the VMs, the user just needs to access the head node in order to submit jobs, which will be later executed in the worker nodes.

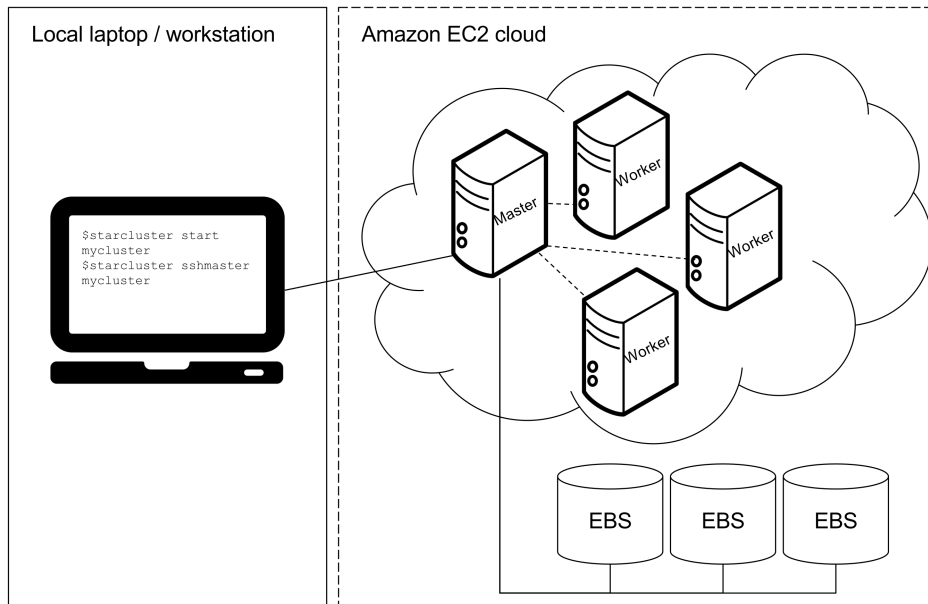


Figure 2.11: Architecture of the Cloud Scheduler auto-scaling strategy.

Dynamic TORQUE [104] is also a suite of Python scripts but this time running alongside TORQUE resource manager and being integrated with an OpenStack cloud infrastructure. It has two operation modes: active mode (see Figure 2.12) and passive mode (see Figure 2.13). In the active mode, the Python scripts actively query the TORQUE job queue, creating new instances when having idle jobs in the queue, and deleting them when no longer required or under “not responding” status reported by the TORQUE manager. In the passive mode, instead of instantiating new nodes and adding them to TORQUE, TORQUE is configured to interact with a special worker node consisting on a fixed-size pool of VMs. The worker node controlling the pool of VMs uses a custom implementation of the Portable Batch System (PBS) Machine Oriented Mini-server (MOM) service, which apart from communicating with the PBS server, manages the set of VMs being used.

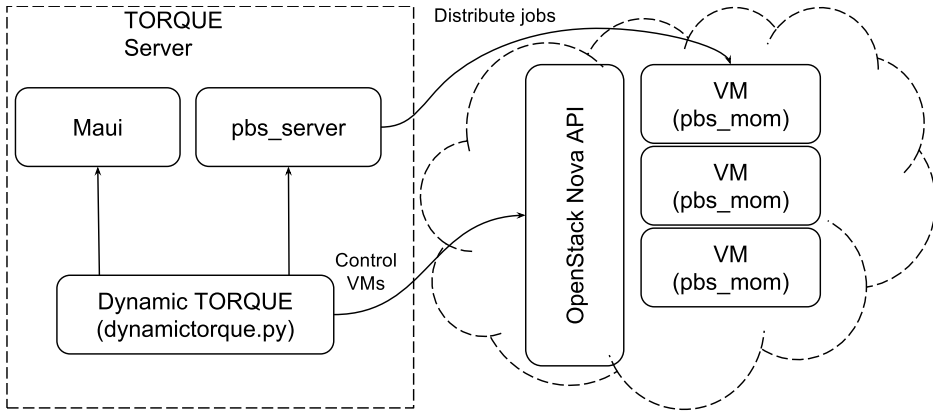


Figure 2.12: Architecture of the Dynamic TORQUE auto-scaling strategy working in active mode.

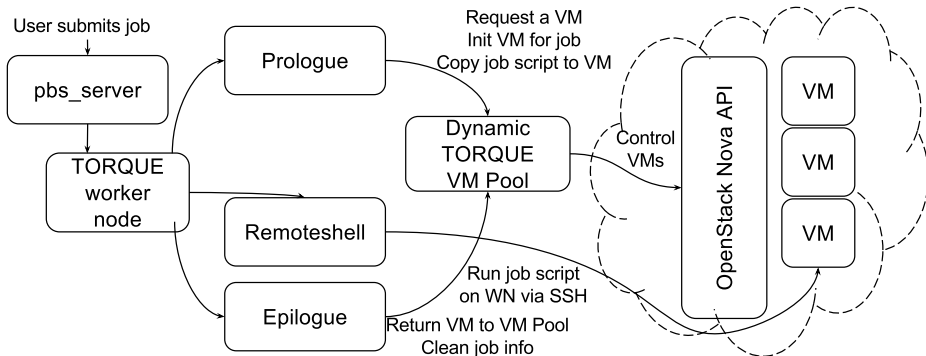


Figure 2.13: Architecture of the Dynamic TORQUE auto-scaling strategy working in passive mode.

In general, the described existing solutions lack flexibility in terms of both the possible cloud solutions and distributed resource managers, they can be used with. Additionally, in StarCluster the number of worker nodes does not vary based on the workload, what in some circumstances will lead to an inefficient use of the resources and in other occasions to undesirable waiting time of the jobs in the queue. The efficient way Dynamic TORQUE is managing OpenStack cloud resources, attracted our interest in using it as the base of our auto-scaling strategy (see Section 4.2), improving it.

## 2.6. Workflows scheduling in the cloud

The workflow paradigm appeared long before the birth of cloud computing. As mentioned, since the appearance of cloud computing, several workflow scheduling techniques have been proposed. Most of them based on ideas adopted from the techniques developed for the grid computing environment, but of course taking into account the specifics of the cloud environment such as the higher flexibility to adapt the infrastructure to the faced workload. Although there exists a significant number of techniques, there are still some common open issues which need to be corrected. For example, most of the techniques are working in a simulated cloud environment (i.e. cloudsims [16]) instead of a real one. This fact, first does not provide the users with a real infrastructure where to run their workflows; and second, reduces the complexity of the cloud environment to a given number of simulated features, what can in turn produce differences with a real environment. Another common problem is that some techniques are taking into account only one type of flavour (i.e. number of CPUs, amount of Random Access Memory (RAM), etc.) for the instantiated virtual machines.

The following summarises the most representative solutions of the state-of-the-art culminating in a table containing the common and specific limitations of the to be described approaches (see Table 2.1). More detailed analyses of current workflow scheduling techniques in the cloud can be found in [77, 101, 2].

The BAR scheduling algorithm [46] performs the scheduling based on three factors, namely the data locality, network status and computational load of the system. The scheduling is split in two phases: an initial assignation and a later iterative refinement by reassigning tasks.

In [38] the authors designed a scheduling algorithm making use not only of cloud resources but also of other computing platforms. The scheduling in this case is also performed in two steps. The first step consists on an *a priori* analysis of the tasks affinity to the different platforms, whereas the second one performs an heuristic-based scheduling to assign the platform. In turn, the platform assignation could be either static, using the task affinity analysis, or dynamic with different configurable criteria.

The scheduling technique described in [27] follows a two-level tasks scheduling mechanism, which balance the workload based on VM utilisation. In addition, it considers the VM assignation to physical resources. If during execution a physical host is overloaded, the VM is moved to another one.

[43] describes a deadline-based scheduling algorithm split in two phases. The



first stage determines the latest time each of the tasks should start in order to meet the deadline. In the second step, a reasoner module calculates the amount of required resources based on historical data of previous executions.

The work described in [57] considers cost and deadline constraints to dynamically provision resources in IaaS clouds for scientific inter-related sets of workflows. This work proposes 3 different scheduling strategies: a static one based on the estimated execution time, considering the time and cost constraints; a dynamic one without admission, where the cost constraint could be violated; and a dynamic one with admission, which checks if the cost constraint will be violated before executing the workflow.

In [94] the authors propose 4 different scheduling strategies: 2 static and 2 dynamic. The first static algorithm adds a new worker if a user-defined threshold is not surpassed, if surpassed it assigns the job to the first worker expected to be free. Instead, the second static algorithm does not consider limitations on the number of workers. The third algorithm –and first dynamic one– takes into account the execution history and also predicts the workload based on the workflow specification. The last algorithm consists of a dynamic assignation of tasks to the workers with the better performance/cost ratio.

In the last considered algorithm in this document [70], the aim of the authors is to provide a robust scheduling by modelling possible system failures and performance differences. In addition to robustness, they are also considering the performance and cost metrics. Based on the three mentioned metrics, the authors propose 3 different strategies. The first one maximises the robustness while reducing the cost and execution time, in the mentioned order. The second one interchanges the last two metrics prioritising the time more than the cost. The last one corresponds to a weighting system where the end-users can weight on their preference the 3 previously mentioned factors.

Problem	Algorithm						
	[46]	[38]	[27]	[43]	[57]	[94]	[70]
Consider only tasks of the same duration	■						
Set of uniform tasks (CPU, I/O), translating into idle CPUs in the case of I/O intensive applications					■		■
Simulated environment (cloudsim)	■	■	■			■	■
One VM per task (deleted after the task is finished)			■				
Deadline-based scheduling (not taking into account performance or cost)				■			
Only one type of instance flavour				■	■		
New machines instantiated based on historical data (not specified what happens when historical data is not available)				■		■	
Limited to a fixed set of programs						■	
Only one task per VM can be run						■	
New VM even if the task is small						■	
Only executes one workflow at a time						■	
VMs could be used by tasks coming from the same workflow but not from others						■	
Scheduling algorithm complexity							■

Table 2.1: Limitations of the mentioned workflows scheduling algorithms. A black cell indicates that the algorithm of the given column is facing the problem given row.

# 3 Infrastructure

---

In this chapter the devised cloud computing based solution is described. The different components of the infrastructure are explained in separated sections. Section 3.1 outlines the used IaaS cloud computing middleware. The user authentication mechanism, which extends the basic authentication of OpenStack using LDAP is described in Section 3.2. Section 3.3 first describes the data management components of OpenStack for virtual machine images, volumes and object containers. Second, it outlines the underlying distributed file system of the whole infrastructure, which sets no single-point of failure. In addition, the mentioned section describes the grid computing technologies used to extend the original functionality of OpenStack (i.e. GridFTP and Globus Online). The computing part of the setup is described in Section 3.4. This section includes basic information about the used virtualization technology and it additionally describes how the TORQUE distributed resources manager, a RESTful Web Services front-end and the Galaxy workflows management system have been integrated into the system. Section 3.5 describes the network setup of the solution. The Horizon web-based management portal of OpenStack is briefly described in Section 3.6. This chapter concludes with the explanation of how the different parts of the system have been interconnected in Section 3.7.

## 3.1. Cloud computing solution: OpenStack

A core target of this work is the efficient computation of scientific applications on top of cloud infrastructures, addressing the related problems. From the range of available cloud service models, IaaS represents the most suitable one due to its broad range of configuration possibilities. In addition, results of a

performed analysis [49] comparing different PaaS and IaaS solutions suggest that IaaS solutions have a better performance for HPC applications.

From the wide range of available IaaS solutions (e.g. Amazon Web Services, Microsoft Azure, OpenNebula), we chose an already available OpenStack cloud computing middleware due to its uptake and flexibility [69]. OpenStack has received a significant amount of uptake and development over the last years compared to existing solutions such as OpenNebula. Additionally, regarding performance OpenStack represents the most stable solution [93]. Even being an open source project enjoys significant industry support of companies like IBM<sup>1</sup>. As the rest of the IaaS offerings, OpenStack is composed by several components that can be separately deployed. These components are (see Figure 3.1):

- Authentication
- Compute
- Data Storage
- Imaging service
- Networking
- Dashboard

---

<sup>1</sup><http://www-03.ibm.com/press/us/en/pressrelease/43892.wss>

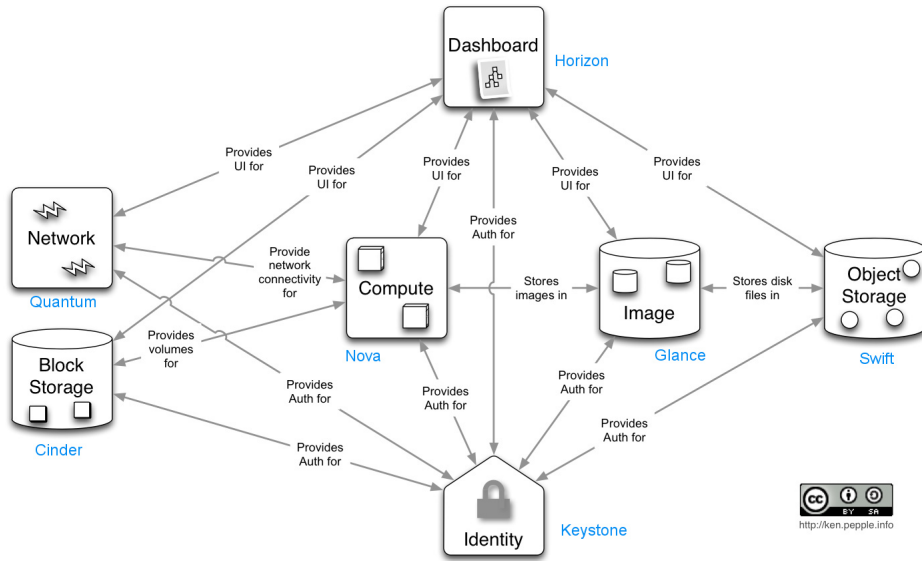


Figure 3.1: OpenStack components.

The different components of OpenStack could be accessed through RESTful web service calls. These calls may originate from the Horizon Dashboard interface of OpenStack or from command-line tools depending on the use case and user preference. Horizon represents the main interface to the cloud computing middleware. It is a user-friendly web-based graphical user interface, which is described in more detail in Section 3.6.

Besides the external Web Service calls, OpenStack components communicate through the RabbitMQ<sup>2</sup> queuing system. The queuing nature of RabbitMQ enables OpenStack to process in order rapidly arriving requests even when the arrival rate surpasses the processing capabilities, a situation where RabbitMQ uses a queue served in FCFS order.

## 3.2. Authentication

The authentication component of OpenStack is called Keystone. The regular way to access it is calling a web service, which authenticates the user given a pair of username and password. After a successful authentication, the user receives a

<sup>2</sup><http://www.rabbitmq.com>

token from the Keystone component, which is subsequently used to authenticate requests to the rest of components.

Similarly to UNIX systems, OpenStack has also the concept of user groups. These groups, termed tenants in OpenStack, typically represent organisations. This arrangement of users into tenants enables OpenStack-based cloud infrastructure providers to offer resources to separated sets of users, who are not aware of using the same physical resources. This follows a similar line to the public cloud offerings.

The Keystone component provides different options for storing the user credentials as well as tenants. The default option is a database backend, but it can also work with a LDAP-based backend. Our OpenStack deployment has been configured to work with the LDAP-backend. Alternative federated authentication and authorisation methods such as OpenID and OAuth were considered in the initial design stages of the system. However, LDAP was selected due its better integration with the different system components, its simpler configuration and yet sufficient number of authentication features for this thesis. This has limited the Federated Identity Management (FIM) brought by OpenID and OAuth.

Several components developed/used in this thesis require authentication mechanisms, such as the Galaxy workflow management system. Therefore, a core functionality for managing the security of the system is a central authentication mechanism. LDAP-based authentication mechanisms were already implemented in Grid computing infrastructures [31] to enable secure and coordinated resource sharing between different groups. Also in grids, users of the same organisation are grouped together conforming what is known as Virtual Organisations (VOs) [32]. In VOs access rights associated with the membership within specific VOs are of a coarse granularity. Within OpenStack, VOs are mapped onto the tenants, thus enabling the participants of one project to easily share resources among themselves.

However, although grid computing already implements LDAP-based authentication mechanisms, easy and flexible user management was still a problem. To overcome this issue, we are using DirGrid, a LDAP-based grid/cloud account management developed at the RISC Software GmbH company. DirGrid enables a more flexible and easier management of users and their access rights via a graphical interface compared to a manual modification using the command-line. This system is being applied to manage the access rights to resources like the cloud infrastructure, but also to the workflow manager and applications beyond the presented in this work.

The grid computing concept of VOs is used to assign the users to specific

OpenStack tenants, therefore facilitating the management of access rights. A good example is given by the access to the local GridFTP [3] server, enabling users to access the service and to use Globus Online to transfer data to the local OpenStack containers. All the access rights based on the VOs are represented within the central LDAP tree of the installation, which is subsequently queried by the different services during user authentication. It is worth mentioning that DirGrid stores the access privileges in multiple LDAP trees, which are fully controlled by their owners, thus allowing a distributed management.

The user LDAP tree holds references to the resources, which are accessible to the user group which maintains it, while the LDAP trees holding the information about the resources maintain a list of authorised users in order to enforce access control locally. These two types of LDAP trees as well as the references between them are maintained by the DirGrid module to ensure that flexible *ad hoc* resource sharing can be offered without relying on some type of centralised management. When changes in access rights are performed by authorised resource owners the updates of the access rights as well as the notification of affected users takes place immediately, thus a user is able to see which resources have available at any point in time.

DirGrid was built using LDAP, X.509 certificates as well as Secure Sockets Layer (SSL) [33] for encryption. The security of the DirGrid system is maintained through SSL encryption of all network connections and by having all users as well as servers authenticating themselves through their X.509 certificates.

As already mentioned, we use DirGrid to manage users and tenants (being equivalent to user groups or projects) to which they belong. The access rights to the different services implemented in this thesis are also maintained by the DirGrid middleware and represented in a local LDAP tree from where the user authentication is queried, when a user logs in to a system.

Using a Web browser, users can access a web-based interface which enables them to handle the modification of access rights stored in the LDAP tree in a user-friendly manner. Depending on his role the user can either modify the access rights of users to the local resources (as admin) or manage ones private contacts and their information which they have been sharing with him, as an ordinary user.

Secure Shell (SSH) keys have been used to provide user authentication to the VMs hosting the different components of the system. In addition, they have been used to enable password-less secure authentication between the VMs taking part of the TORQUE execution cluster. OpenStack, similarly to other IaaS solutions, enable the creation of SSH keys and their assignment to specific cloud instances.

### 3.3. Data management

OpenStack separates its storage system into different services:

- Glance for managing the images
- Cinder for storage volumes
- Swift for object containers

Glance provides the core functionality for starting new instances. The images or templates these instances are based on can be configured beforehand by the OpenStack system administrator or also by non-administrative users. Glance is not considered a pure storage component because it is not storing the images, it just stores meta-data about such images. On the other hand, Cinder provides storage volumes, which are typically attached to already running instances. These volumes appear to the guest operating system of the instance as additional block devices. They need to be partitioned and formatted before they can serve as persistent data storage to instances. Once this process is terminated, the volume can be reattached to a new instance.

Glance and Cinder address the management of block storage including virtual machine images and storage volumes. Instead, Swift is used to access object stores, enabling the storage of files independent of a specific running instance. As other components of OpenStack, Swift containers can be accessed either through the Horizon web dashboard or through Web Service calls. In this work, we have integrated Swift with GridFTP enabling direct mass data transfers to/from the Swift containers as described in Section 3.3.3.

#### 3.3.1. Underlying file system (Ceph)

The underlying storage system of the used cloud setup is Ceph [97]. Ceph is a distributed file system able to run on commodity hardware and running on top of the Linux operating system. Its pseudo-random data distribution mechanism (CRUSH [98]) determines data locations of Ceph clients without querying a central node. As a consequence, Ceph does not contain a centralised meta-data server, therefore it does not have a single point of failure. We decided on Ceph, based on the better fault tolerance and the limited speed and high latency of the Swift default built-in OpenStack file system.

When data needs to be read or written, Ceph groups data blocks into the so-called placement groups. These placement groups are internally represented



in a hash table, where the object name is hashed, resulting in a placement group identifier. This placement group identifier points to a primary storage server in the Ceph ecosystem. The location of this server can be found by calling the CRUSH function on the client side. Clients then connect to the identified server and stores its objects there. Any replication of the object is done within the Ceph storage nodes, without requiring intervention of the client. In fact clients ignore where their objects have been replicated, they just call the CRUSH function to obtain the endpoint where the data is stored.

On top of the distributed object storage service offered by Ceph runs a service called RADOS Gateway<sup>3</sup>. This gateway enables users to interface with Ceph objects via Hypertext Transfer Protocol (HTTP) commands. Additionally, it allows users to use the Swift [79] as well as the Amazon Simple Storage Service (S3) APIs<sup>4</sup> by linking the RADOS Gateway with the OpenStack installation as performed within the presented solution (see Figure 3.2).

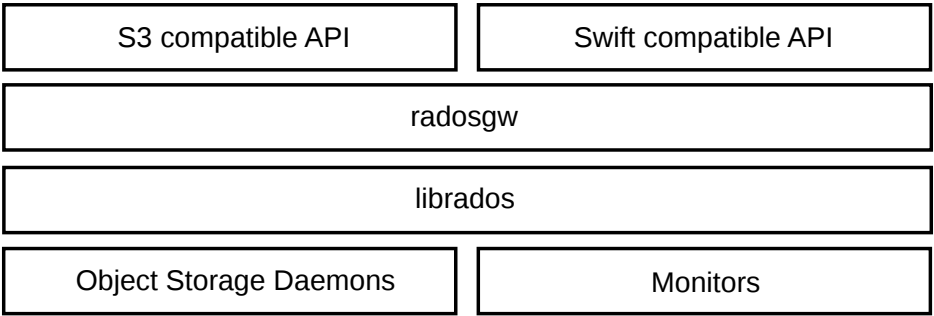


Figure 3.2: Ceph Object Gateway. The Amazon S3 and OpenStack Swift APIs running on top of the RADOS Gateway daemon.

3.3.2. Globus Online

The potentially large input and output data sizes to be managed in the devised cloud infrastructure make it essential to use a well-established protocol for reliably and securely transferring large datasets. For this purpose, GridFTP was configured, which in turn enables the usage of the data transfer mechanism of the Globus Online (GO) initiative [4, 29]. Compared to FTP or SFTP, GridFTP provides a public key security mechanism, parallel Transmission Con-

<sup>3</sup><http://eu.ceph.com/docs/wip-3060/radosgw/>

<sup>4</sup><http://aws.amazon.com/s3/>

trol Protocol (TCP) streams, striping, partial file transfers, and with GO easy to install software for client-side, long-running and asynchronous data transfers. GO only monitors and controls the transfer, while GridFTP manages the actual data transfer between two GridFTP servers registered as GO endpoints. The GridFTP endpoint present in the cloud environment of this thesis can access the Swift object storage through Portable Operating System Intefarce (POSIX) calls. This is possible because the Swift containers are mounted into the local file system of the endpoint via a CloudFuse<sup>5</sup> daemon.

### 3.3.3. Endpoint Setup

#### GridFTP User authentication

To provide direct access to the OpenStack storage system through GridFTP, a certain server setup is required. The core modules used in this setup consist of several Globus Toolkit services depending on the use of short-lived or long-lived credentials and the integration to Globus Online:

- GridFTP: the GridFTP service implementation facilitates fast and reliable data transfers between GridFTP endpoints. To ensure the security of the data transfers, the GridFTP and MyProxy [66] components are used to enable secure authentication towards the system as GridFTP is based on authentication through proxy certificates. The advantage of using proxy certificates is that an entity is allowed to act securely on behalf of another entity, as long as the proxy certificate is valid [89].
- MyProxy: if a user is accessing the server through a GridFTP transfer agent, the transfer agent receives the proxy certificate. This proxy certificate can either be created for long-lived or short-lived credentials. In the case of a long-lived credential, the user becomes a personal certificate signed by a national certification authority. In case of a short-lived credential, MyProxy generates a temporary certificate, which authorises the transfer agent on behalf of the user to manage his or her data transfers.
- Simple Certification Authority (CA): These temporary certificates allows the GridFTP transfer agent to take care of data transfers between servers as long as they are valid, without the need of storing the user password. The proxy certificates are generated by MyProxy, which instructs the Simple CA<sup>6</sup> to generate and sign the proxy certificates.

<sup>5</sup><https://github.com/redbo/cloudfuse>

<sup>6</sup><http://simpleauthority.com/>

- Extended Internet Daemon (xinetd): acts as a open source super-server daemon to automatically start the GridFTP and MyProxy servers. Various configuration parameters of such servers are defined in their start-up scripts<sup>7</sup>. These parameters include listening ports, port ranges, and application-specific values such as user certificates directory.

The previously described authentication mechanism has been integrated into the cloud environment used in this work. The first step to achieve this integration was the replacement of the OpenStack standard keystone user directory by LDAP as explained in Section 3.2. The virtual machine hosting the GridFTP endpoint for the object storage system looks up for users and passwords using standard Pluggable Authentication Module (PAM) [76] mechanisms and connecting to the central service. As MyProxy is using PAM for user authentication, the integration of LDAP for enabling short-lived credentials is trivial.

## Object Storage Integration

After successfully integrating the LDAP and MyProxy authentication mechanisms with the used cloud environment, the next step is the interconnection of the storage system. To allow GridFTP managing data, the Swift-compatible object storage of OpenStack should be made directly accessible. Since the GridFTP service is only able to operate with a local file system, the cloud-based object storage needs to be mounted into the local file system.

A daemon using CloudFuse<sup>8</sup> is used to perform this task. This daemon running in background manages the CloudFuse processes required to mount the storage. To simplify the mounting process from the user point of view, as soon as a user has authenticated successfully on the system, a LDAP script module communicates with the daemon through a UNIX domain socket. The daemon reacts on the socket input and starts a script to mount all the object containers the user is able to access into its home directory, if no CloudFuse processes are running for the user so far. A separate background thread of the daemon stops the CloudFuse processes after a defined timeout. Figure 3.3 shows the workflow starting at the endpoint activation up to the object storage mount.

---

<sup>7</sup>[http://wordpress.risc-software.at/en-austriangrid/?page\\_id=422#xinetdkonfig](http://wordpress.risc-software.at/en-austriangrid/?page_id=422#xinetdkonfig)

<sup>8</sup><https://github.com/redbo/cloudfuse>

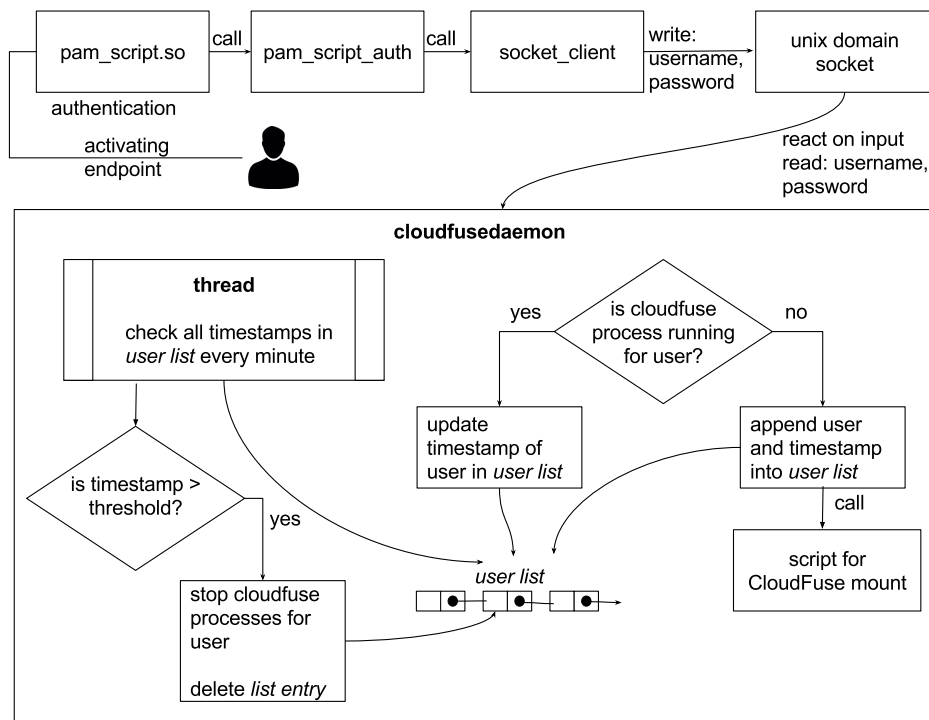


Figure 3.3: Interaction of all components to mount the users corresponding containers of the object storage.

## Endpoint Registration

To allow data transfer using the Globus initiative tools, the GridFTP and MyProxy services have to be registered as endpoints at the Globus Online file transfer agent. This can be done with the web service provided by Globus Online or with the mobile client GOTransfer<sup>9</sup>.

The information required for such registration is the address of the GridFTP server as well as the address of the MyProxy server. Additionally, the subject's Distinguished Name of the certificates is required to communicate with both services. After this registration, the endpoint can be activated and used to transfer data through the Globus Online file transfer agent.

<sup>9</sup><http://www.risc-software.at/de/aktuelles/newsaevents/179-advanced-computing-technologies/841-android-app-gotransfer>

## Architecture and Workflow

With the described setup the architecture shown in Figure 3.4 is established. Whenever the user activates an endpoint, a username and password have to be provided. These credentials are sent within an authentication request to the endpoint server to activate. The MyProxy component is contacted, later it sends an authentication request to PAM, and finally as soon as the authentication was successful, the object storage corresponding to the user is mounted by the CloudFuse daemon, as explained in detail in Section 3.3.3. MyProxy also sends a certificate request to the CA and returns the created proxy certificate to the GridFTP transfer agent using short-lived credentials. This certificate is needed for data transfers to or from the endpoint. After verifying the proxy certificate, the data can be transferred from or to the endpoint using GridFTP.

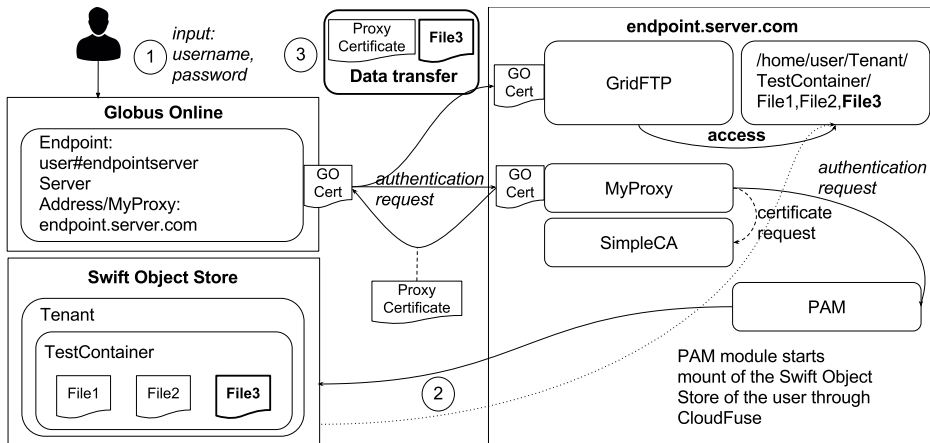


Figure 3.4: Architecture of the Endpoint Setup including the object storage integration.

## 3.4. Computation

Nova is the main part of OpenStack representing its compute service component. This component manages and automates pools of computing resources and is able to work with different virtualization technologies. Nova is able to

work with Kernel-based Virtual Machine (KVM)<sup>10</sup>, VMware<sup>11</sup> or Xen<sup>12</sup> as underlying virtualization technology. As many other OpenStack components, it is written in Python and uses several external libraries such as Eventlet for concurrent programming, Kombu for Advanced Message Queuing Protocol (AMQP) communication and SQLAlchemy for database access. The architecture of Nova is designed to allow the easy horizontal scaling on commodity hardware without specific requirements on proprietary software.

Glance is another important compute component of OpenStack. This component represents the OpenStack Image Service providing discovery, registration, and delivery for disk and server images. It can be used to store and catalogue an unlimited number of images and backups, which might be in turn used as templates. Glance could be seen both as a compute and as a storage module, as it does not store the images, instead it stores the associated meta-data independently of the used storage backend. The default storage backend of OpenStack is Swift, however any other backend able to communicate using the Glance API could be used. In fact, in this thesis the Ceph distributed file system is used to store the server images as explained in Section 3.3.1.

Working together, the Nova and Glance components manage the execution of virtual machines also commonly referred as instances in the cloud computing field. When the Nova compute service receives a request for starting a new instance it also receives the specification of the flavour to be used (i.e. number of CPUs, RAM and disk size) as well as the image to be started for that request. The image contains the file system to be used as the root partition. The service subsequently invokes the underlying virtualization infrastructure (KVM in our case) to boot up the virtual instance.

### 3.4.1. TORQUE distributed resources manager

A number of distributed resource managers exist, being TORQUE<sup>13</sup> a popular and widely used open source solution to manage High Throughput Computing (HTC) clusters. From the usability point of view, both end-users and administrators are familiar with its job submission language and configuration respectively, which are very similar to the PBS [41], what make it adoption easier. From the computational point of view, TORQUE is able to handle large clusters with tens of thousands of nodes and jobs, and large jobs eventually spanning hundreds of

---

<sup>10</sup><http://www.linux-kvm.org>

<sup>11</sup><http://www.vmware.com/>

<sup>12</sup><http://www.xenproject.org>

<sup>13</sup><http://www.clusterresources.com/products/torque/>

thousands of processors. Besides, TORQUE has built-in scheduling algorithms and it is also able to communicate with external schedulers such as Maui (i.e. the one used in this work) and Moab through a clearly defined scheduling interface. We considered Simple Linux Utility for Resource Management (SLURM) [102] in the initial stages, but it uses a fixed scheduler, which although allows changing some parameters, it does not allow to change job priorities and therefore the tasks' execution order.

However, TORQUE has currently no official built-in mechanisms or plugins to interact with any cloud computing solution. Dynamic TORQUE (see Section 4.2) was introduced as an approach to overcome this limitation by having an external component running alongside TORQUE and communicating with the OpenStack API to manage the instantiation and deletion of VMs. Although other solutions targeting specific cloud infrastructures exist in the state-of-the-art (see Section 4.2), the scalability and configuration possibilities of TORQUE as well as its popularity, attracted our attention to integrate it with the described OpenStack cloud installation.

### 3.4.2. RESTful Web Services front-end

A common RESTful Web Services front-end has been designed to allow executing tools in different cloud computing platforms. This common interface simplifies invoking diverse services and interconnecting them to conform a workflow. The designed front-end has been implemented in different programming languages: C# (for Windows Azure) and Java (for other IaaS type of clouds such as Amazon EC2, OpenStack, etc.). This interface provides operations to submit new jobs, cancel previous ones, poll for status, and retrieve intermediate and final results (see Figure 3.5).

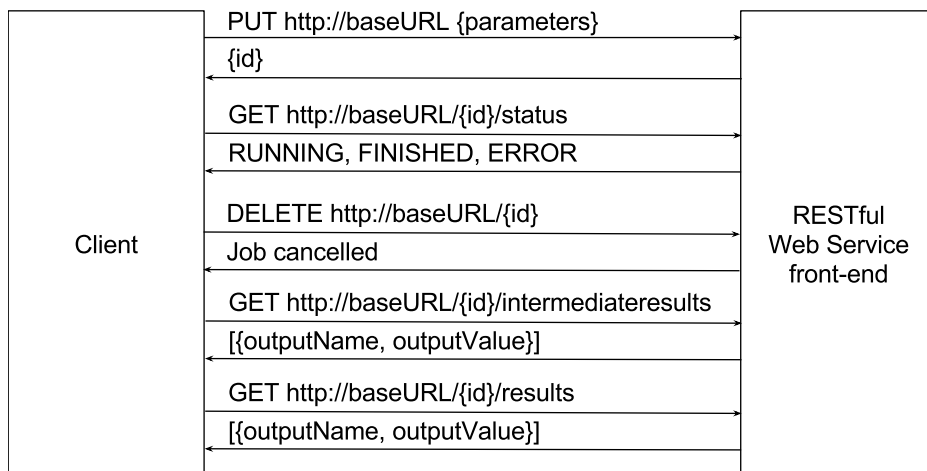


Figure 3.5: Overview of the operations implemented in the designed RESTful Web Services front-end.

To submit a new job the user has to fill the required parameters and data references to files already uploaded to the cloud storage. Once the front-end receives the job submission, it contacts the job scheduler (located within the same instance) to schedule and dispatch the new task. The scheduler and its accompanying auto-scaling strategy will decide whether new instances are required or not as explained in the next chapter, Section 4.2.

At the end of the job submission, the front-end will return a new resource to the user (addressable via a unique URL). This resource can be cancelled, and polled for status, intermediate and final results upon termination. The intermediate and final results will be data references to the actual data present in the cloud storage following the same approach as for the input data. At this point in time, the user can choose either to download the data from the storage or submit the resulting data references as input to another tools. Therefore, these call-by-reference invocations greatly facilitate invocation of a series of services because the data is already available on the cloud infrastructure for a subsequent service without requiring to download and upload it again.

A client-side component has been developed to invoke services implementing the designed front-end. This component has been included as an execution *worker* in MAPI. The main target of MAPI is harmonising the different Web Services meta-data stored in different catalogues. Besides Web Services meta-data unification, MAPI allows invoking services implementing different communica-



tion protocols. To achieve this point, MAPI includes an expandable execution module, which includes various *workers*, each in charge of executing a given service type. The previously mentioned worker extends the already available ones. The developed *worker* enables all the software clients built on top of MAPI such as jORCA or mORCA to invoke the new type of services. An overview of the different components involved in the execution of services following the designed front-end is provided in Figure 3.6.

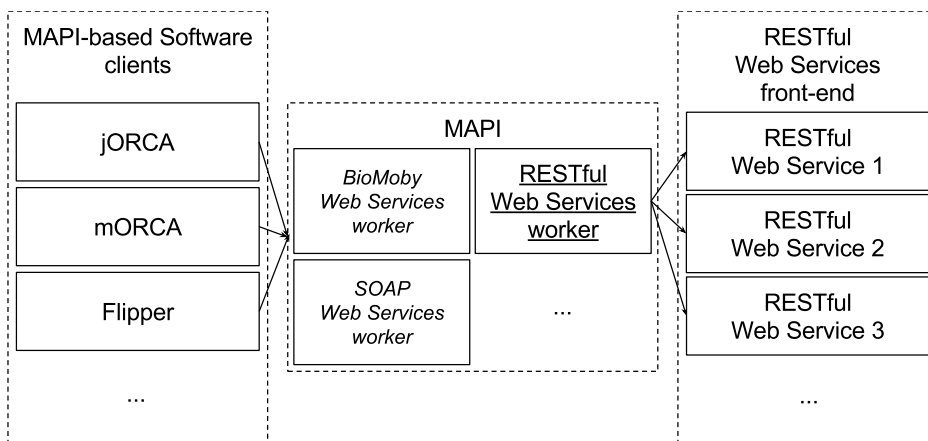


Figure 3.6: Overview of the components involved in the execution of a RESTful Web Service.

### 3.4.3. Galaxy workflows management system

Galaxy [1] is a workflow management system that enables the definition and sharing of scientific workflows. It was originally developed to deal with biological data, however it is currently used in a number of different public servers<sup>14</sup> of different research domains. One of its main objectives is making the management of workflows easier to scientists, which not necessarily have computer programming experience.

It is a web-based application, which implements the core functionality of workflows management systems explained in the background chapter, Section 2.3.1. The main feature is the creation of workflows out of existing or custom modules which are defined by their functionality as well as by input and output

<sup>14</sup><https://wiki.galaxyproject.org/PublicGalaxyServers>

parameters. Users can interactively combine these modules into workflows by interconnecting them on a graphical canvas. These workflows can subsequently be stored for reuse and shared with other researchers, what partially addresses the issue of reproducible research.

Before choosing Galaxy, we also considered other workflow management systems (i.e. e-Science Central [42] and jBPM [34]). However, in e-Science central the infrastructure management should be done manually and the workflows definition could be only performed via the e-Science Central web application. In the case of jBPM, there is not a direct cloud support for such manager, the infrastructure management should be done manually as well, and the workflows definition should be performed with external frameworks/tools such as Eclipse or jBPM designer.

Figure 3.7 shows part of the setup of workflow representing the real-world application GECKO, which is described in Section 5.3. The workflow takes two biological sequences as input. In the first part of the workflow a dictionary of words of a given length is calculated. The second part of the workflow calculates the sequences alignment based on the previously calculated dictionaries.

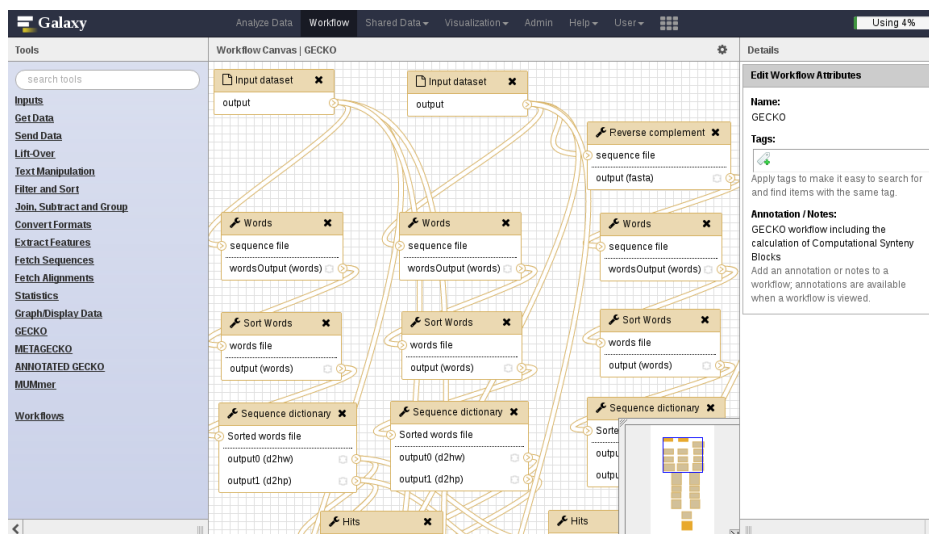


Figure 3.7: Part of the GECKO workflow shown in the Galaxy workflow editing canvas.

Figure 3.8 shows the setup of the workflow, which enables to set specific

parameters for the modules such as files for the input channels of the initial modules in the workflow.

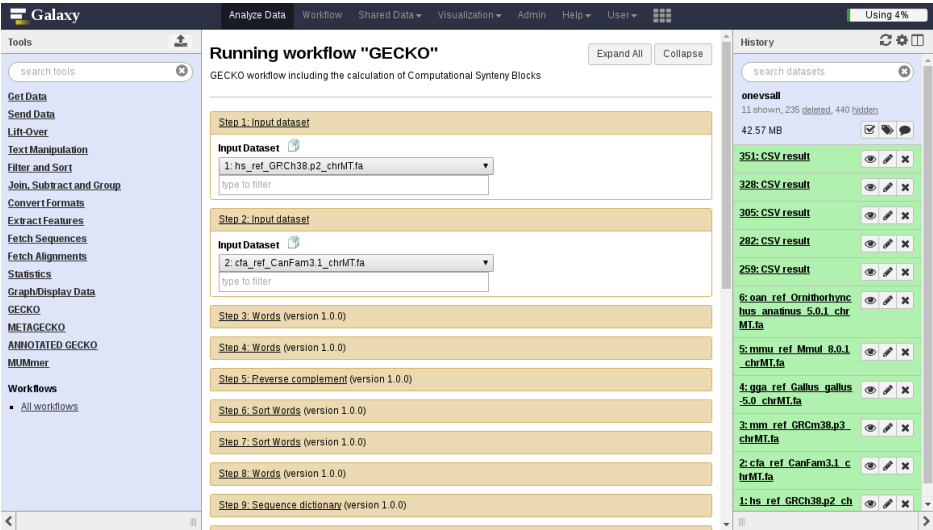


Figure 3.8: Invoking interface of GECKO workflow in Galaxy, showing the required parameters to run it.

Figure 3.9 shows a view during the execution of the workflow. After the execution has finished the user can access the intermediate and final results using the history column on the right side.

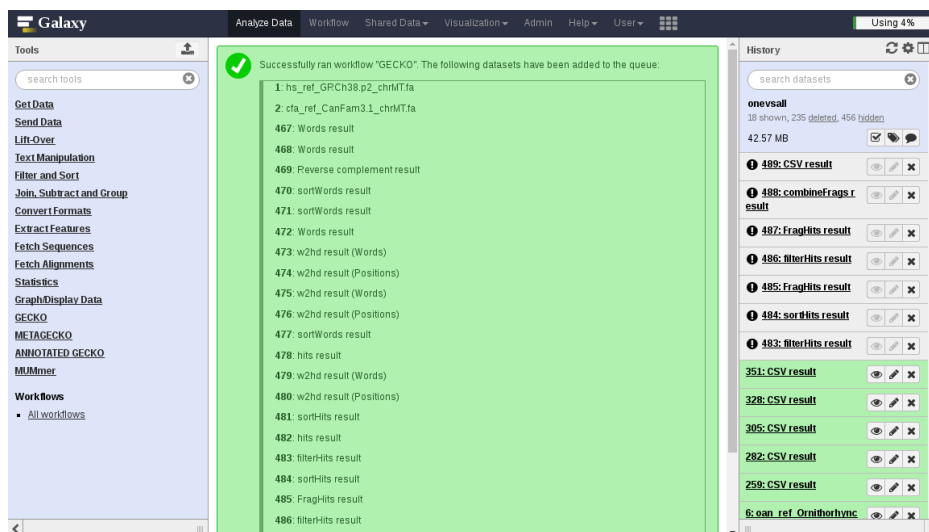


Figure 3.9: GECKO workflow running in Galaxy, already finished tools are shown in green in the right panel, running tools in yellow and waiting tasks in grey.

## Galaxy deployment

The current Galaxy deployment is made using OpenStack Heat templates for the virtual machine configuration<sup>15</sup>. This template defines the required OpenStack parameters to instantiate a machine with Galaxy automatically installed and ready to be used. The template is performing the following steps:

1. Package prerequisites installation and basic instance configuration.
2. Download and placement of the latest stable release of Galaxy from its Git repository.
3. Galaxy configuration files modification.
4. Download Galaxy custom Custom Style Sheets (CSS) files replacing the original ones.
5. Adding Galaxy to be run during the machine start up.
6. Making Galaxy available by modifying the *nginx* server configuration.

<sup>15</sup>[https://svn.mrsybiomath.eu/svn/infrastructure/trunk/OpenStack/Heat/galaxy\\_nginx\\_postgres\\_2\\_servers.yaml](https://svn.mrsybiomath.eu/svn/infrastructure/trunk/OpenStack/Heat/galaxy_nginx_postgres_2_servers.yaml)

Since new analysis tools can be developed and it would be interesting to include them in Galaxy, it is important to separate the tools repository from the Galaxy application itself in order to avoid the re-installation of such tools in case of an instance failure or with the use of a new one. To overcome this problem, we use what is called Galaxy ToolShed, where the admins can define software packages that can be later on included and installed in several Galaxy instances with almost no effort.

## 3.5. Networking

The networking aspects of virtual machine management are addressed by the Neutron component, which is highly configurable to allow integrating virtual machines into a given organisational networking setup. While internal IP addresses are assigned automatically, OpenStack allows the user to assign floating public IPs to running instances.

Neutron consists also of a database for persistent storage. It can be extended with any number of plug-in agents to provide other services such as interfacing with Linux networking mechanisms, external devices, or Software-Defined Networking (SDN) controllers. OpenStack Networking is entirely standalone and can be deployed to a dedicated host. In the presented cloud environment, the Neutron component has been deployed to a centralised management host.

The performed networking deployment uses the Modular Layer 2 (ML2) plug-in of OpenStack to communicate with Linux bridges. This allows regular (non-privileged) users to manage virtual networks within a project and additionally includes the following components:

- Project (tenant) networks. Project networks provide connectivity to instances for a particular project. Non-privileged users can manage project networks based on the user privileges an administrator defines for them. Project networks use Generic Routing Encapsulation (GRE)-tunnels [39] and VM tagging to allow having multiple separate networks generally using private IP address ranges [73]. These internal networks lack connectivity to external networks such as the Internet.
- External networks. This component provides connectivity to external networks such as the Internet. Only administrators or operators can manage external networks because they interface with the physical network infrastructure. From the point of view of non-privileged users, they just have a

list of available public IP addresses with Internet access.

- Routers. Routers connect project and external networks. Only privileged users can create new routers to interconnect different projects or to provide access to an external network to a particular project. By default they implement Source Network Address Translation (SNAT) to provide outbound external connectivity and Destination Network Address Translation (DNAT) to provide inbound connectivity on project networks.

Figure 3.10 shows the physical networking architecture of the performed deployment in the used cloud environment. The virtual architecture of the network is shown in Figure 3.11.

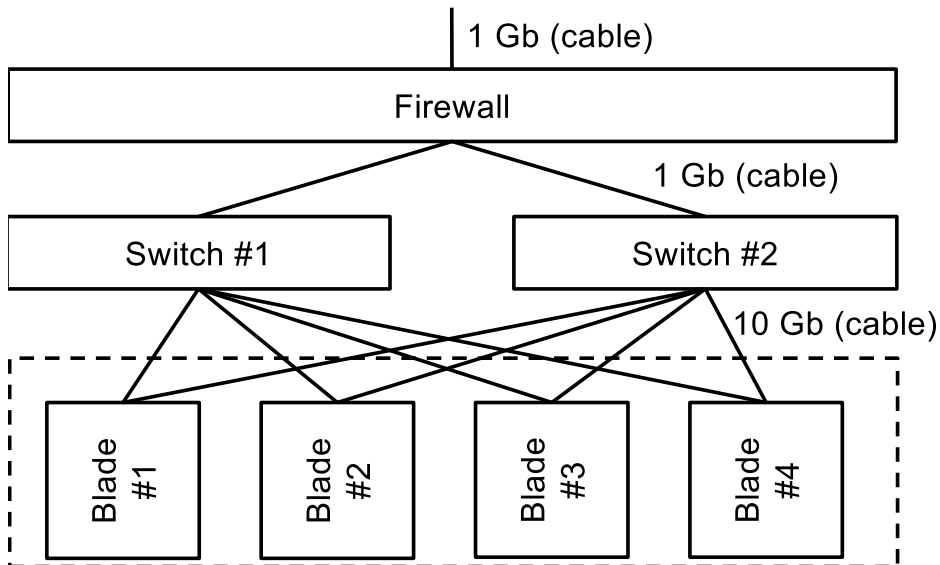


Figure 3.10: Physical architecture of the networking component of the used cloud environment. First there is a physical firewall interfacing with the Internet. Second, this firewall is redundantly connected to two BNC switches. Finally, all the blades of the IBM BladeCenter H have one physical connection to each of the switches.

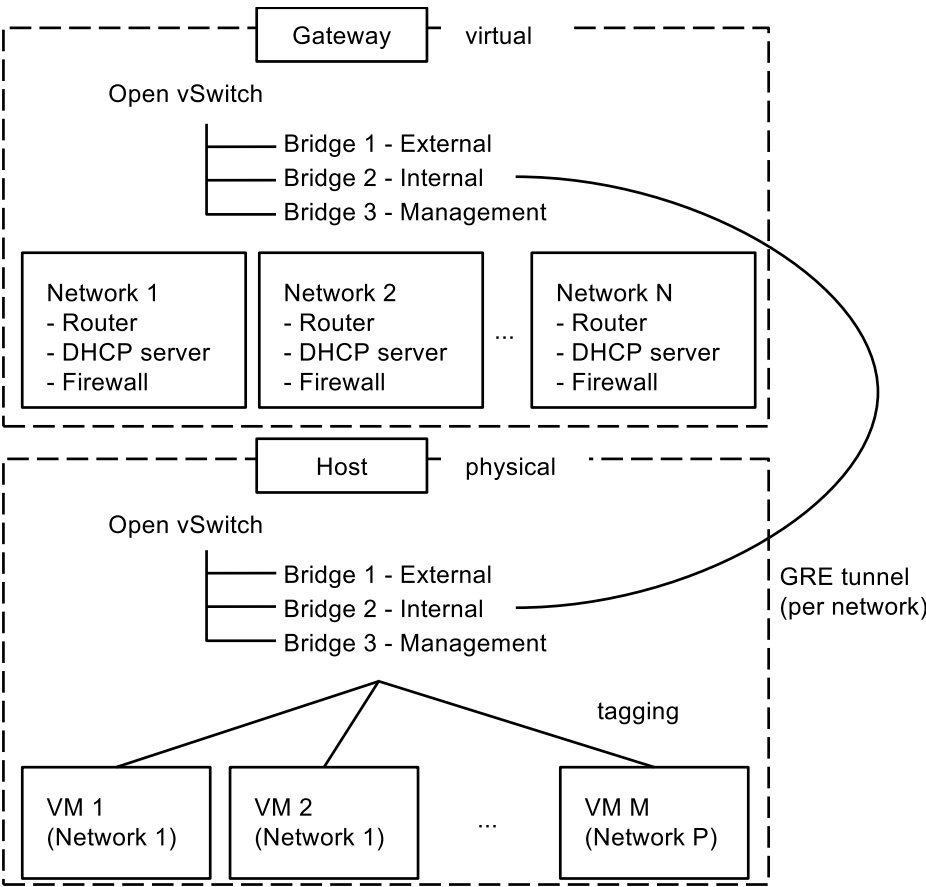


Figure 3.11: Virtual architecture of the networking component of the used cloud environment. There is a virtual central gateway, which runs three Open vSwitch bridges corresponding to each of the networks of the setup (i.e. external, internal and management). Besides, for every network defined in the neutron component, a virtual router, Dynamic Host Configuration Protocol (DHCP) server and firewall are running. The VMs running on each physical host are tagged to a network, and following the same three-bridges structure, their network traffic is tunnelled to the corresponding network bridge using GRE tunnels.

### 3.6. OpenStack Horizon

The OpenStack system offers the web-based user interface Horizon (see Figure 3.12) for administrative tasks as well as self-service management of their virtual machines for ordinary users. It supports operations on virtual instances such as launching, stopping, terminating as well as creating snapshots of them for later use. In addition, volumes can be created, subsequently mounted into the virtual machines and will persist after the virtual machine has been terminated. Other features include the assignment of dynamic (floating) public IP addresses to running virtual instances.

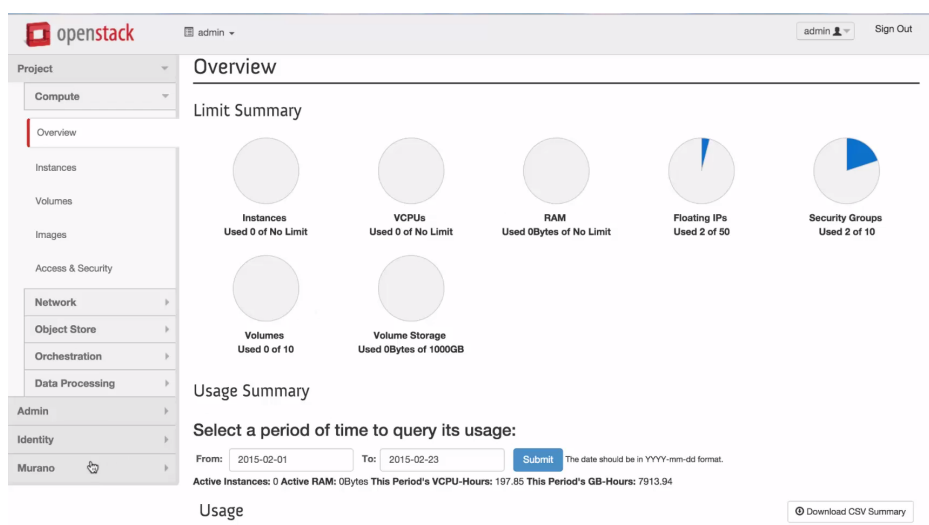


Figure 3.12: Overview page in Horizon

### 3.7. Interconnection between components

The deployment of a cloud computing-based infrastructure, containing tools ranging from command-line to web-based applications, could be performed following different strategies. Traditional deployment approaches of web applications follow a monolithic scheme with one or two servers, which are typically a web and a database server. However, to take benefit from the cloud computing features and to achieve scalability and robustness, more distributed deployment approaches should be used [20].



In the following, we describe how the different components used and developed in this work have been deployed to conform a scalable system architecture running on top of an OpenStack cloud environment. In a first step, to decide the scaling needs of the different components, we categorise them either as asynchronous or synchronous services based on usage properties and expected response time.

Interactive or synchronous services should provide an immediate response to incoming connections. In our case, such services are the ones provided by Galaxy: user login, data analysis, workflows creation/edition, etc. Thus, the main requirement for the user interface is short response time to any of these requests. In a high utilisation scenario, dynamic and swift scale-out of the involved components has to be provided to maintain the response time in the desirable range.

Asynchronous services are services of which users do not expect an immediate response time. In our cloud-based HPC infrastructure these services include computationally-intensive applications and long-lasting data transfers. Additionally, workflows typically represent asynchronous services. Although the infrastructure components belonging to this category: the workflow engine, the job manager, and the GridFTP server; do not need immediate scaling, they are scaled under given circumstances as explained in next chapter.

In a second step we define different storage categories with diverse access patterns and speed requirements. These features influence the data management in the system, and what scaling strategies can be applied in each case. Elements of the used OpenStack cloud environment enables managing data storage, access and the scaling of the devised infrastructure. The different storage categories are the following:

- An *in-Memory storage* is used for data objects which have to be delivered with little response time. For instance, we are using a caching strategy for the static content of the web front-end. This reduces disk access and therefore the response time when users are accessing the system.
- A *database* is used for storing diverse small dynamic data objects describing workflows, execution histories, or user data. A traditional PostgreSQL database management system is used in this case. However, for efficient scaling in other cloud environments, automatically redundant and scaled Database-as-a-Service (DBaaS) offerings, which are included in several IaaS solutions, should be used.
- A *temporary file storage* is used where random access to files is necessary and for applications working with files too large to be stored in the main memory. In the presented architecture, this temporary file storage is used

for data which is currently being processed or transferred to/from the system.

- A *persistent file storage* is used for the rest of the files, storing them in the cloud specific object store. The object store is used as persistent and scalable data storage without the need for scaling or adding virtual hard disks.

The setup guidelines of Galaxy describes the possibility of deploying it on a single computer. However, as identified in previous paragraphs, a distributed deployment is required in order to take profit of the scaling feature of the underlying platform. The performed deployment (see Figure 3.13) interconnects the system components with six virtual machines as starting point, later on this group of VMs is automatically scaled:

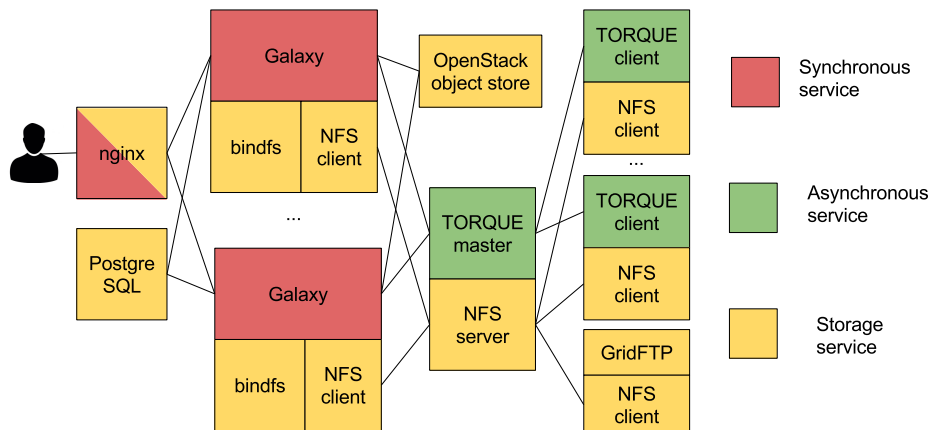


Figure 3.13: Overview of how the different components used in the infrastructure relate with each other. A gateway and several Galaxy application servers as synchronous services, a distributed execution cluster and a GridFTP server as asynchronous services, and a database server (PostgreSQL), a Network File System (NFS) share and an object store as storage services.

- A *gateway server* represents the primary contact point between users and the infrastructure. In the performed setup the nginx<sup>16</sup> HTTP/HTTPS proxy is serving as both load-balancer and web server. Additionally, it is responsible for the in-memory storage for serving static content. Scaling this

<sup>16</sup><http://nginx.org/>

server is only possible in combination with further load-balancing in front of this service or using Domain Name System (DNS) load-balancing [15].

- A *galaxy server* provides the user interface of the workflow management system as well as a Galaxy ToolShed. When required, new Galaxy VMs are included to continue providing synchronous replies. In such scenario, the load-balancing between VMs is performed by nginx, data is shared through a common NFS share, and concurrent access to the database is managed by PostgreSQL.
- A *database server*, required by Galaxy, is installed on a separate machine. This separation enables multiple Galaxy instances using the same server in a high utilisation scenario. On cloud solutions with DBaaS solutions, they should replace this server.
- A machine containing a *NFS server and the TORQUE master node* is included. The NFS server coordinates the temporary file storage, whilst the TORQUE master node performs the task scheduling and distribution. This machine is the central node for asynchronous services and therefore the weakest point of the current setup as it represents a single point of failure.
- A *GridFTP server* is installed in another VM. Data delivered to the end-point served by this GridFTP server is stored on the temporary storage provided by the machine hosting the NFS server. Users can migrate files uploaded with GridFTP to the persistent file storage via the Galaxy web interface. Although GridFTP scaling strategies are well documented in [3], in our current setup they are not considered.
- *TORQUE worker node(s)* represent the last part of the infrastructure. One or several VMs corresponding with the worker nodes subsequently queried by the TORQUE master conform this group. It is worth noting that the number of active virtual machines in the setup is dynamically changed by the presented auto-scaling strategy, which is explained in more detail in the next chapter, Section 4.2.

The integration of the different components has been described and automated using an OpenStack Heat template<sup>17</sup>. Such template for deployment automation defines the required OpenStack parameters to instantiate a set of VMs and installs the described system within minutes. In addition, the deployment task could

<sup>17</sup>[https://svn.mrsybiomath.eu/svn/infrastructure/trunk/OpenStack/Heat/galaxy\\_nginx\\_postgres\\_2\\_servers.yaml](https://svn.mrsybiomath.eu/svn/infrastructure/trunk/OpenStack/Heat/galaxy_nginx_postgres_2_servers.yaml)

be performed also using other automation tools like like Chef<sup>18</sup>, puppet<sup>19</sup> or ansible<sup>20</sup>.

---

<sup>18</sup><https://www.chef.io/>

<sup>19</sup><https://puppetlabs.com/>

<sup>20</sup><https://www.ansible.com/>

# 4 Scheduling and auto-scaling

---

In this chapter the schedulers used in conjunction with the TORQUE distributed resources manager are described in Section 4.1 (a basic background on tasks scheduling is available in Appendix B). First, the mentioned section explains the built-in FCFS scheduler of TORQUE, and second, it describes the configuration of the Maui scheduler, how the job priorities are calculated based on different factors, and the nodes allocation policy for such jobs, which is not part of the scheduling itself but of the process management. Section 4.2 describes the developed auto-scaling strategy, its configuration parameters and the implemented scaling decision mechanisms. The section and chapter finalise with the description of the deployment scenario.

## 4.1. Scheduling

The default TORQUE scheduler implements a variety of common scheduling algorithms including round-robin, FCFS, and fair-share. Each of these can be further configured to indicate what class of job should be run first as for example shortest or largest walltime. Nevertheless, these default schedulers become useless in big infrastructures because of their performance and little configuration possibilities. TORQUE offers a clearly-defined scheduling interface to allow integrating custom schedulers such as Maui or Moab, which might provide greater configuration options and performance compared to the built-in scheduling algorithms.

### 4.1.1. Built-in TORQUE FCFS scheduler

Although several schedulers are included by default in TORQUE, FCFS is the default out-of-the-box scheduler configured. This algorithm, as its own name indicates, schedules tasks as they enter the system. Making an analogy with a priority-based scheduler, this algorithm only prioritises the time a job has been in the waiting queue. The rest of possible job priority parameters (see Section 4.1.2) are not considered. Therefore, the scheduling decision is very simple, what could compromise the performance of the system. In general terms, its performance is enough for small distributed environments with simple workloads. However, for bigger environments with more complex workloads more sophisticated schedulers such as Maui are required.

### 4.1.2. Maui scheduler

The Maui scheduler appears as a reasonable alternative to tackle the limitations of the built-in TORQUE schedulers. The limitations are reduced by allowing a fine-grained configuration of the job priority to decide the next job to run. In addition, the node allocation parameters to decide in which node(s) the job will run can be configured. The scheduler behaviour depending on these parameters is evaluated in Section 5.3.

#### Job priority parameters

Maui permits to dynamically calculate the job priority at each scheduler iteration based on the weighting of a number of factors<sup>1</sup>. These factors are broken down into a two-level hierarchy of priority factors and sub-factors each of which can be independently assigned a certain weight. The first-level parameters have been assigned to zero when not interested in such group of factors and to one when interested. The previous balances the importance of the groups of factors, and propagates the actual priority calculation to the sub-factors. The sub-factors or parameters we are considering in this thesis are:

- the total requested *walltime*, which defines a hard clock-time limit<sup>2</sup> between a task is entering to execute till its completion (this time does not include the waiting time in the queue since it is *a priori* unknown by the user). This user-defined value indicates approximately for how long the job will

<sup>1</sup><http://docs.adaptivecomputing.com/maui/5.1.2priorityfactors.php>

<sup>2</sup>jobs reaching this hard limit will be removed from execution

be executing. Typically in batch systems, long-running jobs are assigned a lower priority, since a little bit longer waiting time in the queue will not be significant compared to the actual execution time and its effect in other jobs is notable, reducing their average waiting time in the queue.

- the *queue\_time* representing the time the job has been queued. This dynamic value, periodically updated by the scheduler, provides a measure of the system fairness. In a fair system the standard deviation of the waiting times of the different jobs should not be too high.
- the *expansion\_factor*, calculated as the division of the time in the queue by the *walltime*. It has a similar effect to the *queue\_time* factor but favours shorter jobs based on their requested *walltime*. To prevent this factor to grow inordinately, a configuration parameter determines the minimal *walltime* value to be used in the denominator of the fraction.
- the number of *nodes*, *cores* and amount of *memory* (i.e. the requested job resources). These three sub-factors will determine whether jobs asking for a large amount of resources would obtain a higher priority or if such priority would be for jobs asking for a small amount of resources.

Equation 4.1 shows how the job priority is calculated. The values of the different weighting factors have been set up to prioritise jobs in the following order: short jobs (giving  $w$  a negative value); jobs queued for a long time (providing a small positive value to  $q$ ); short-medium-sized jobs waiting for a long period of time (giving  $e$  a positive value bigger than  $q$ ). In case of equal job priority values at this point, the job requesting less resources will have a higher priority (giving small negative values to the  $n$ ,  $c$  and  $m$  factors).

$$\begin{aligned}
 job\_priority = & \mathbf{w} * walltime + \mathbf{q} * queue\_time + \mathbf{e} * expansion\_factor \\
 & + \mathbf{n} * nodes + \mathbf{c} * cores + \mathbf{m} * memory
 \end{aligned} \tag{4.1}$$

### Node allocation policy

At a second stage, after deciding which job to run, the node or nodes where this job will actually run should be selected. In the presented infrastructure (see Chapter 3), this decision is even of higher importance than in conventional HPC environments. In a cloud-based computing environment, this decision will determine the cost of the execution. First, the amount of money to be paid depends on the selected type of machine, which has a given instance flavour.

Second, the decision will determine the machines that can be freed up because of not being used anymore.

In the original scheduler of TORQUE the node allocation policy is not configurable, instead, Maui provides several options to choose based on user preference. It has a set of predefined configuration choices such as CPU load, first available, fastest nodes, nodes with the minimal number of resources, etc. Additionally, it provides a fully configurable allocation policy, which enables selecting the execution node(s) based on a range of factors.

The allocation policy used in this thesis prioritises nodes at several levels aiming to reduce the cost by deleting as soon as possible dynamic nodes and using the nodes with the best-fitting amount of resources. Considering this, we have devised the following three-level nodes priority calculation:

- the first level prioritises static nodes over dynamic nodes. TORQUE, neither Maui, are aware of the type of nodes. However, a given priority value could be assigned to a node while instantiating it. In our case, we have assigned a higher priority to static nodes. The reason is that static nodes are always running, what produces a given cost, and therefore the system should use them before dynamic ones.
- the second level prioritises the nodes based on the amount of resources they have. This level is relevant when two given dynamic nodes have the same priority in the first level. In such situation, we prioritise the node with a lower number of resources but still fitting the job. This will later allow the auto-scaling strategy deleting dynamic nodes with a big amount of resources, for which usually a larger amount of money should be paid.
- the third level assigns a higher priority to the nodes most historically used. This level is important when the second level has selected dynamic nodes. In such scenario, selecting the most historically used dynamic nodes will leave free the less used dynamic nodes, which will possibly be removed later on by the auto-scaling strategy.

In summary, the node allocation policy is aiming at having a compact yet effective set of computing nodes. Keeping it compact with the help of the auto-scaling policy will reduce the cost and make the system management easier.



## Backfill

Backfill is a scheduling optimisation which allows making a better use of the available resources by an out-of-order execution of the jobs. Maui orders the jobs into a ‘high priority first’ sorted list based on the priority factors explained in Section 4.1.2. Then it starts distributing jobs one by one until it reaches a job that cannot be started. All jobs have a *walltime* limit, therefore Maui can determine the latest completion time of all jobs. Consequently, Maui can also determine the earliest the needed resources will become available for the highest priority job.

Backfill uses this information to calculate what is known as *backfill windows*. These windows represent time frames a given set of nodes are idle between the finishing time of the currently running job and the ‘earliest job start’ of the higher priority job. Enabling backfill allows the scheduler to start other lower-priority jobs in such windows as long as they do not delay the highest priority job. In any case, Maui creates a job reservation of the required resources for the highest priority job at the appropriate time.

Backfill offers significant scheduler improvement. In a typical large system, the system utilisation could be increased by 20% [45], and the job turnaround time, defined as the time between a job enters the system until its execution is finalised, by an even greater amount. Because of the way it works, backfill tends to clearly improve small and short jobs, and moderately improve larger ones.

The performance improvement of backfill comes at a price. There exist several minor drawbacks. In first place, it diminishes the job prioritisation a site has chosen, because short lower-priority jobs might enter execution before higher priority tasks. Secondly, only the start time of the  $n$  highest priority job is protected by a reservation, therefore tasks from the  $n + 1$  highest priority one onward could be delayed. The third problem is that backfill assumes users are performing a good estimation of how long their tasks will be running. In negative cases, such as the example provided in the next paragraph, the start of some high priority jobs could be delayed.

Consider a scenario involving a two-processor cluster. Job A has a 4 hour walltime and requires 1 processor. It started 1 hour ago and will reach its walltime in 3 more hours. Job B is the highest priority idle job and requires 2 processors for 1 hour. Job C is the next highest priority job and requires 1 processor for 2 hours. Maui examines the jobs and correctly determines that job A must finish in 3 hours and thus, the earliest job B can start is in 3 hours. Maui also determines that job C can start and finish in less than this amount of time. Consequently,

Maui starts job C on the idle processor. One hour later, job A completes early. Apparently, the user overestimated the amount of time his job would need by a few hours. Since job B is now the highest priority job, it should be able to run. However, job C, a lower priority job was started an hour ago and the resources needed for job B are not available. This results in a higher priority job being delayed. The described scenario is illustrated in Figure 4.1.

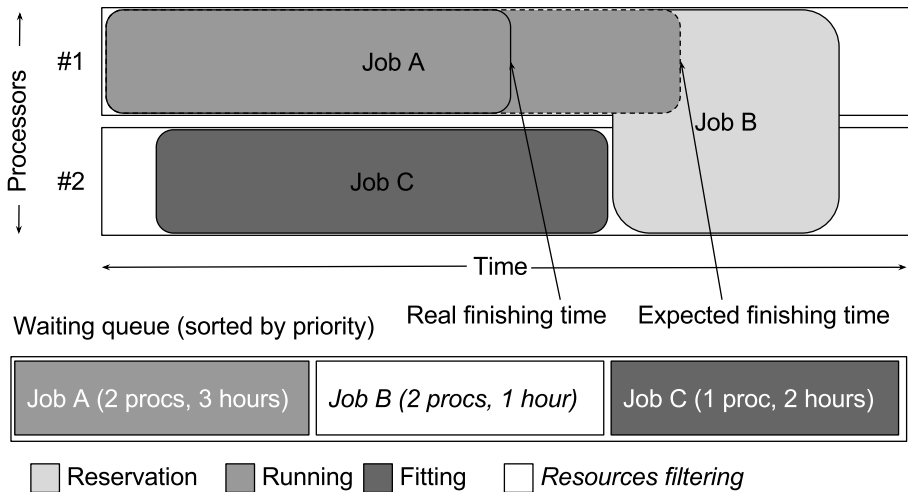


Figure 4.1: Example scenario of job delays caused by backfill.

Although there do exist some minor drawbacks with backfill, its net performance impact on the workload of a site is very positive. Even though a few of the highest priority jobs may get temporarily delayed, studies have shown that only a very small fraction of jobs are truly delayed and when they are, it is only by a small fraction of their total queue time. At the same time, many jobs are started significantly earlier than would have occurred without backfill.

If backfill is enabled, Maui allows using three different algorithms: *first-fit*, *best-fit* and *greedy*. In this thesis we have chosen the *best-fit* mechanism which is defined in Algorithm 1 and illustrated in Figure 4.2. The best-fit algorithm increases resources utilisation compared to *first-fit*. The *greedy* algorithm has been discarded because it may induce delays of higher priority jobs.

---

**Algorithm 1** Best-fit backfill algorithm
 

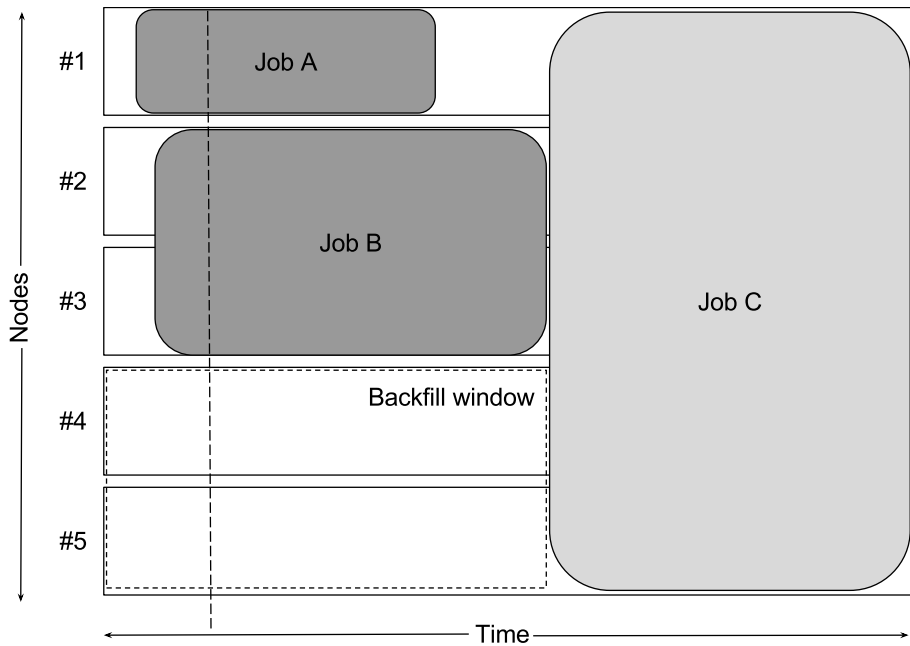
---

```

1: repeat
2:    $filteredJobs \leftarrow filter(jobs, backfillWindow)$ 
3:   for all  $job$  in  $filteredJobs$  do
4:      $job.degreeOfFit \leftarrow job.numberOfProcessors$ 
5:    $sortedJobs \leftarrow sortByDegreeOfFit(filteredJobs)$ 
6:    $start(sortedJobs[1])$  ▷ Start the best fitting job
7: until  $length(filteredJobs) = 0 \parallel length(idleResources) = 0$ 

```

---



Waiting queue (sorted by priority)



Figure 4.2: Example of the best-fit backfill algorithm in an artificial scenario. Jobs A and B are running, and there is a reservation for Job C after Job B finishes its execution. The highest priority job in the queue (Job F) needs 3 nodes to run, this creates a backfill window in nodes 4 and 5. This backfill window is filled with the job with the highest number of nodes fitting in such window in number of request nodes and wall time (in this case Job G).

## 4.2. Auto-scaling strategy

The developed auto-scaling strategy is based in the Dynamic TORQUE [104] solution, which consists on a suite of Python scripts running alongside the TORQUE

resource manager and being integrated with an OpenStack cloud infrastructure. Dynamic TORQUE has two operational modes: active mode and passive mode. In the active mode (i.e. the one being used), Dynamic TORQUE keeps a fixed specified number of static worker nodes, a group that can be extended later by adding dynamic nodes until a user-specified value. Figure 4.3 provides an overview of Dynamic TORQUE running in active mode.

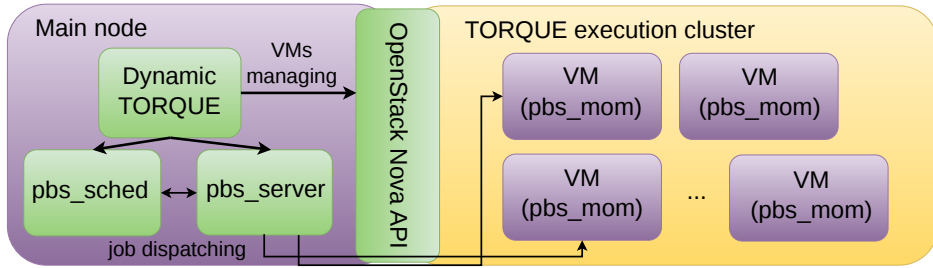


Figure 4.3: Overview of Dynamic TORQUE running in active mode. VMs/nodes are represented in purple rectangles, running services in green rectangles and the yellow rectangle indicates the nodes belonging to the TORQUE execution cluster. All VMs/nodes are running inside the described OpenStack cloud infrastructure.

#### 4.2.1. Configuration parameters

The devised auto-scaling strategy can be easily configured by modifying a single configuration file. This file contains a wide range of parameters, being the following the most representative ones (the first three were already available in the original implementation and the last three have been developed in this thesis):

- **Number of static instances:** This parameter refers to the number of worker nodes that will be instantiated and continuously available overtime while the system is running.
- **Number of dynamic instances:** This second value indicates how many workers can be dynamically added or removed depending on the faced workload. The addition of this and the previous parameter determines the maximum number of running instances. It is worth noting that a machine that was initially supposed to be static can finish its execution much earlier than a dynamic one created afterwards. In this situation, the static instance is deleted and the dynamic one becomes static in order to preserve the correct

number of static instances (see more details about this in the Algorithm 3). This is done with the aim of maintaining each time the minimum number of required resources.

- **Max idle time of a worker:** This configuration parameter refers to the maximum amount of time that a worker can be without executing tasks, or in other words, idle. The value of this parameter will not only change the number of active instances, but will also possibly affect the amount of time required to execute a task or group of tasks.
- **List of flavours:** This parameter indicates the set of instance types that can be used. In the original Dynamic TORQUE implementation, it was only possible to specify one flavour, which in turn is used to instantiate new workers. The original implementation did not take into account the amount of required resources specified during job submission. In our modified implementation we add the possibility to configure a list of possible flavours. One of the flavours will be used as base flavour for jobs not specifying the required amount of resources, whereas the rest will be used in accordance with the job requirements.
- **Billing period:** This configuration parameter indicates the granularity of the billing period of the cloud provider (whether it is per minute or per hour). This value is not particularly significant in private cloud environments, however it is important for public cloud environments. In such environments, the parameter will determine when the node is deleted in order to take the best possible benefit out of the VM until its billing period is reached.
- **Provisioning delays:** This last value indicates the average time each of the flavours take to be provisioned since requested. This parameter will determine whether the strategy decides to add a new worker or not depending on the duration of the task. If the task duration is shorter than the provisioning delay it does not start a new worker node.

#### 4.2.2. Scaling decision mechanism

The developed auto-scaling strategy queries the TORQUE server about the status of the worker nodes in order to determine, either if additional nodes are required, or if idle nodes might be removed. For up-scaling the cluster of workers, the presented strategy follows Algorithm 2. Briefly, the strategy evaluates if it

compensates to add a new node by comparing the addition of the estimated duration of the waiting tasks with the provisioning delay of the required flavour. A new worker is only added in case the estimated duration is greater than the delay (assuming the maximum number of workers has not been reached). Algorithm 3 is followed to down-scale the cluster. Shortly, in this scenario, the presented strategy retrieves a list of idle workers. When a worker of this list has been idle for more than an specified time interval then it is marked to be deleted. It is not directly deleted because the system keeps using it for short tasks until the billing period of such worker is reached.

---

**Algorithm 2** Up-scaling algorithm

---

```

1:  $tasks \leftarrow waitingTasks(queue)$ 
2: for all  $task$  in  $tasks$  do  $d \leftarrow d + t.duration$ 
3:  $f \leftarrow requiredFlavour(tasks)$   $\triangleright$  Minimum required flavour based on the
   specification of the tasks
4: if  $d > f.provisioningDelay$  then
5:    $addWorker(f)$ 

```

---



---

**Algorithm 3** Down-scaling algorithm

---

```

1: procedure DOWN-SCALING
2:    $\triangleright$  First, we need to delete the nodes already marked to be deleted
3:    $tbdNodes \leftarrow toBeDeleted()$ 
4:   for all  $node$  in  $tbdNodes$  do
5:     if  $currentTime - node.billingPeriod < 60$  then  $\triangleright$  60 seconds
6:        $deleteWorker(node)$ 
7:    $\triangleright$  Second, we iterate over the list of nodes and mark new nodes to be
   deleted
8:    $idleNodes \leftarrow idleNodes(nodes)$ 
9:   for all  $node$  in  $idleNodes$  do
10:    if  $node.idleTime > conf.maxIdleTime$  then
11:      if  $isStatic(node)$  then
12:         $dNode \leftarrow dynamicNode(nodes)$ 
13:         $dynamicToStatic(dNode)$ 
14:         $markItToBeDeleted(node)$ 
15:         $node.idleTime \leftarrow node.idleTime + conf.pollingInterval$ 

```

---

### 4.2.3. The deployment scenario

The described auto-scaling strategy has been deployed in the cloud infrastructure described in Chapter 3. The presented deployment contrasts with the one reported in [104], where they allocate the key components in physical machines off the cloud, whereas in our case we are running all the services inside the cloud.

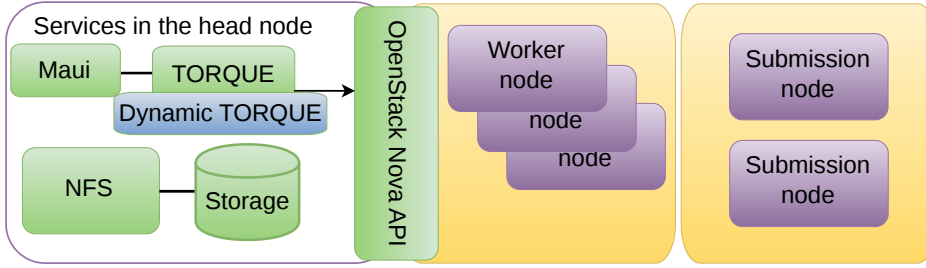


Figure 4.4: Deployment of Dynamic TORQUE in the described cloud infrastructure. In the head node the PBS and NFS servers are running. Additionally, there are two logically separated groups: First, the worker nodes which are used to execute jobs; second, the submission node from where the users send jobs to the TORQUE queue.

Although everything is running inside the cloud, we make an abstract separation in two main groups (see Figure 4.4), one of them composed by the key services (TORQUE and NFS), and the other composed by the worker nodes. The Maui scheduler is deployed to the head node together with TORQUE in order to schedule the jobs present in the TORQUE waiting queue. Additionally, there are some submission nodes from where new jobs are added to the TORQUE queue.

The Dynamic TORQUE service, running in the logical group of the key instances, manages the worker nodes using the Openstack API. It monitors the health of the worker nodes, shutting down inaccessible nodes and firing up new ones to replace them. Dynamic TORQUE also supports to have worker nodes out of the cloud or inside the cloud but not necessarily under its control.

All the worker nodes are launched from a pre-configured VM image specified in the configuration file. This image has the minimum amount of services required to work in the mentioned environment. Since NFS is used as the shared file system, the image has a NFS-client installed and configured to mount the NFS export of the key server. In addition, a PBS MOM service is configured to communicate with the TORQUE server, which will then dispatch it jobs to run.



# 5 Experimental Evaluation

---

In this chapter we present the evaluation of the system. First we introduce the synthetic and real-world workflows used to evaluate the system designed in this thesis (see Section 5.1). Second, Section 5.2 provides an overview of the metrics used to evaluate the performance of the system. Third, this chapter contains the results and discussion of evaluating the system using the selected set of workflows based on the chosen evaluation metrics (see Section 5.3). Finally, the chapter concludes with a summary of the main factors affecting the system behaviour in terms of the scheduling and auto-scaling decisions in Section 5.4.

## 5.1. Workflow applications

We are considering two kinds of workflows to evaluate the devised system: synthetic workflows already used in similar recent works [54, 58, 92] obtained from the workflow generator gallery [19, 21], as well as three illustrative synthetic examples, which have been designed to point out the benefits and limitations of the scheduling strategy. The second type of workflows corresponds to three real-world applications, two of them developed in this work, and the other one coming from our collaborations with the bioinformatics and biomedicine domains scientists of the Mr.Symbiomath project.

### 5.1.1. Synthetic workflows

As previously stated, in order to evaluate the algorithms on a standard set of workflows, we downloaded workflows from the workflow generator gallery. Such gallery contains synthetic workflows derived from structures and parameters of

real-world applications. From all the available workflows we have selected three different ones: Ligo [23], Montage [10] and CyberShake [37] (see Figures 5.1, 5.2 and 5.3 respectively). The selected subset represents a heterogeneous set of synthetic workflows in terms of task duration and computation pattern, including 50 tasks in the Ligo and Cybershake workflows and 100 tasks in Montage. Each of the tasks composing the available workflows has its corresponding execution time extracted from real execution traces. In addition to the execution time, the workflow specification also contains the input and output data sizes so that the I/O load can be at some extent simulated.

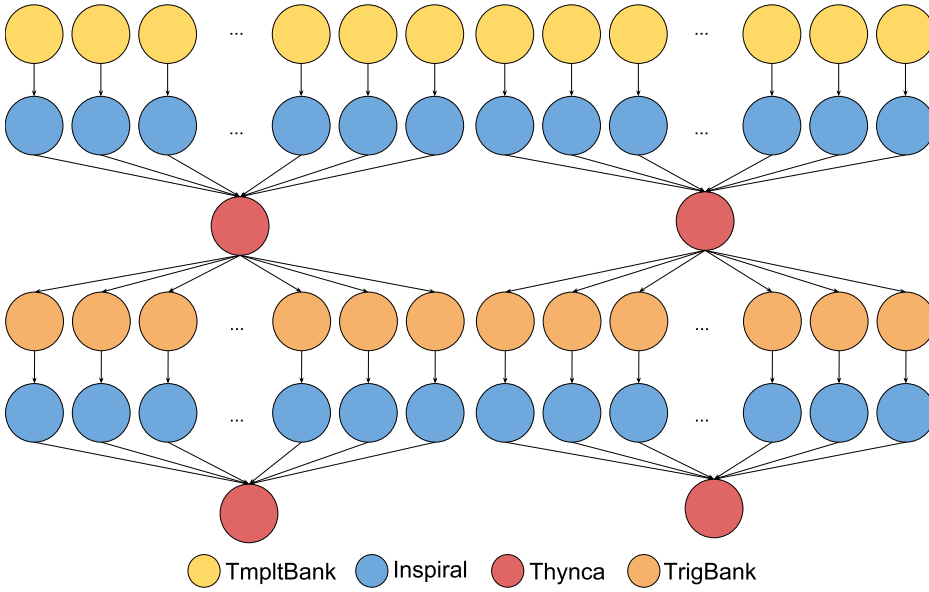


Figure 5.1: Overview of the Ligo workflow. This workflow is composed of 4 different tasks. The first  $n$  tasks of the entry level read the input data and perform some initial simple calculations. Each of the tasks of this level triggers the execution of a task belonging to the second level. In the third level, several reduce tasks concatenate part of the  $n$  partial results generated. Additionally, this task triggers the execution of  $m$  tasks following a similar scenario compared to the first two levels. Once this second group of tasks is correctly executed, the partial results are reduced by several tasks.

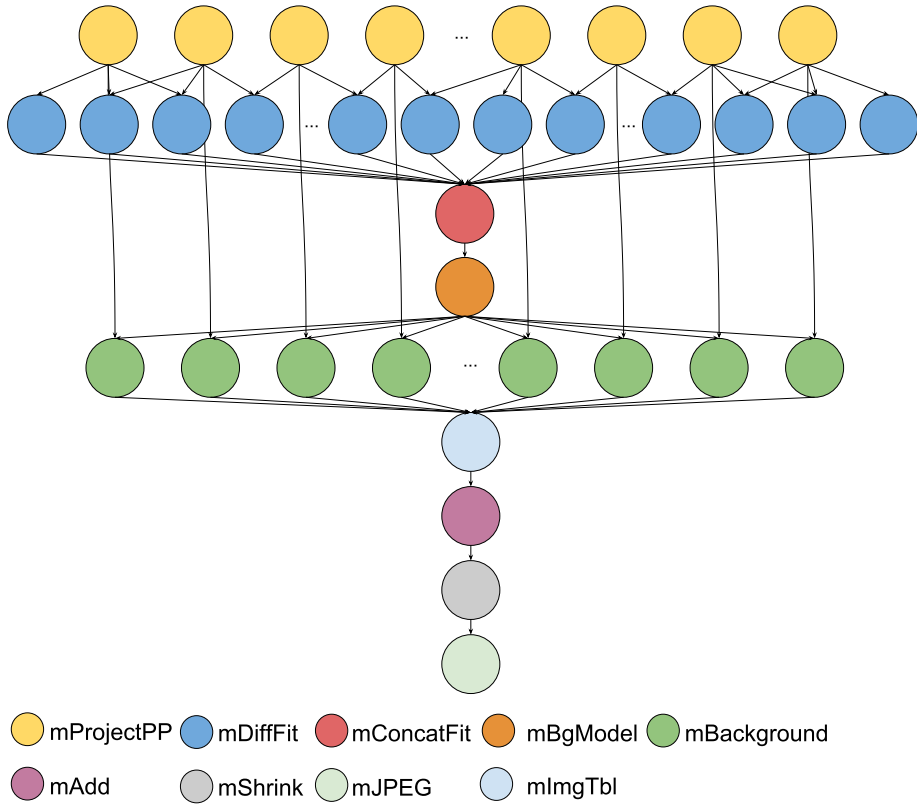


Figure 5.2: Overview of the Montage workflow. This workflow is composed of 9 different applications. The first  $n$  tasks at the workflow entry level read the input data and trigger at their end three tasks at the second level. Once all the tasks of the second level are executed, a reduce task concatenates the partial results. The next task performs some calculation over the concatenated file and splits again the file to be processed by several independent tasks. To finalise, a pipeline of 4 tasks first concatenates the partial results and then performs some calculations until obtaining the final result.

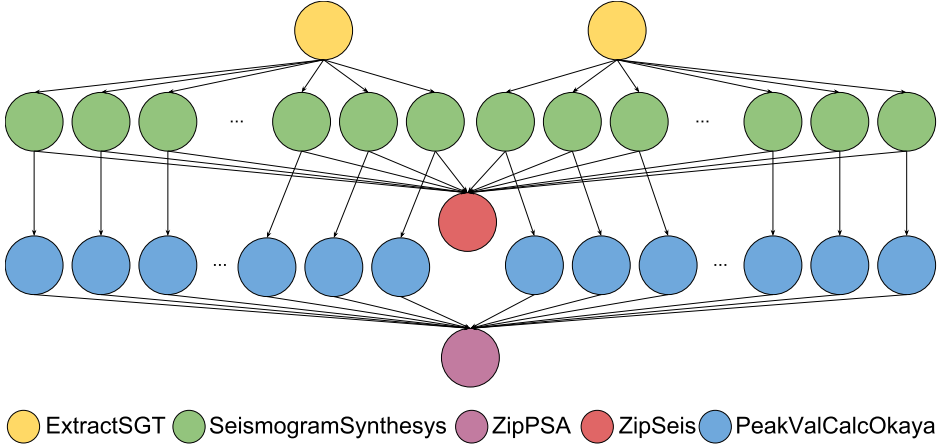


Figure 5.3: Overview of the Cybershake workflow. This workflow is composed of 5 different applications. The first two applications read and split the input data in several chunks to be processed in the second level of the workflow. The finalisation of each of the tasks of the second level triggers the execution of a task in the third level. Once all the tasks of the second and third level finalise their execution, two respective reduce tasks are executed.

Since the binaries nor the input files of the mentioned workflows are available (to the best of our knowledge), we are just simulating their execution in our deployed infrastructure. The simulation covers two aspects: execution time and data reading/writing. For this purpose each of the programs conforming a workflow is wrapped and simulated using the *stress* command available in the *ubuntu* software repositories. This command impose load on a given number of cores, during a specific time and performing a given number of I/O operations (specified in number of bytes to be read and written). Within our simulation, the *stress* command will: first, execute for the time specified in the task description inside the workflow trace file; and second, will read/write the same amount of bytes as also stated in the trace file. To overcome the simulated execution using the *stress* command, we are additionally evaluating the system with real-world workflows.

The 3 selected synthetic workflows represent typical computational patterns/schemes with data splitting/distribution tasks and partial results reduction or synchronisation tasks. For instance, the Ligo synthetic workflow is somewhat similar to the real-world workflow GECKO (see Section 5.1.2), which has two branches to calculate a dictionary of words for the input sequences, and then

a pipeline of 4 tasks to calculate the seed points of the sequence alignment, sort them, filter them, and finally produce the set of shared segments. It is important to note that synchronisation tasks would affect the efficiency of the system with regards to resources utilisation. For that reason, the auto-scaling strategy presented in the previous chapter, scales automatically the number of nodes to the faced workload.

In addition to the workflows obtained from the workflow generator gallery, a small set of illustrative examples has been used to point out the benefits and limitations of the devised scheduling strategy. These examples contain a set of independent tasks with different duration ranging from seconds to a couple of minutes. The first workload (test-1) contains an unbalanced set of tasks for the available resources in the testing infrastructure (see first paragraph of Section 5.3.1 for more details about the infrastructure). In this situation, the devised scheduling strategy is expected to behave differently to the FCFS algorithm as will be discussed later on in Section 5.3. The second example (test-2) contains a diverse set of tasks in terms of duration as well (see Section 5.4.1 for the duration distribution of this and the rest of the examples). In this case, all the long-running tasks are concentrated at the beginning, where the devised scheduling strategy is expected to produce a lower waiting time in the queue compared to the FCFS algorithm because it schedules the execution of the short tasks to the beginning, whilst the FCFS preserves the submission order. The third and final illustrative example (test-3) contains a set of tasks with a similar distribution to the second example, but in this case with the longest running tasks concentrated at the end of the workflow. In this scenario, the waiting time in the queue is expected to be similar for both scheduling algorithms (i.e. FCFS as the baseline, and the strategy devised in this work) because the sorting based on priority performed by the priority-based scheduler would not significantly change the execution order.

### 5.1.2. Real-world workflows

To contrast that the obtained results in the simulation of the synthetic workflows are also valid in non-simulated executions, we selected three real-world workflows coming from the bioinformatics and biomedicine fields. These real-world workflows have been executed using large input data in terms of size and cardinality in order to illustrate the improvements of executing them in the designed auto-scaled execution cluster.

### Pairwise genome comparison workflow

The first real-world workflow corresponds to GECKO [84], which is a pairwise sequence comparison software developed in this thesis. GECKO is a modular application aimed at identifying the similarities shared by a pair of genomes. The final output of this workflow is a collection of High-scoring Segment Pairs (HSPs), which contain the coordinates and quality measures of the genome segments with a certain similarity. The modular design of GECKO allows further comparisons to be performed without the need to recalculate intermediate results, which are stored on disk, and thus without sacrificing performance. The modular design is the following (see also Figure 5.4):

1. One-off creation of a dictionary of words of length  $K$  (commonly referred as K-mers) for each genome or sequence. This dictionary contains the words and their occurrence positions along the sequence.
2. K-mer dictionaries are then used to calculate the starting points (or hits) of possible alignments. These seed points correspond to all possible word matches produced between dictionary words (different  $K$  values can be used at this point to optimise the sensitivity or execution time of the application).
3. To finalise, the application produces the set of HSPs based on the calculated starting points.

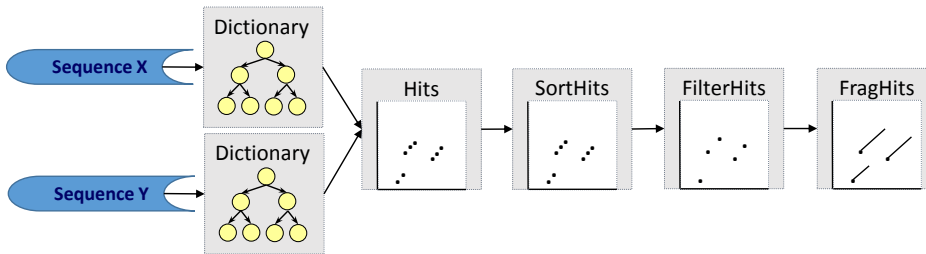


Figure 5.4: The pairwise genome comparison workflow.

Although originally designed for pairwise comparisons, GECKO is also able to efficiently compute multiple genome comparisons given the avoidance of intermediate results recalculation. These massive exercises are especially suitable for this kind of auto-scaled infrastructures, where the amount of resources can be scaled in accordance to the faced workload. Although GECKO is faster than equivalent software, its execution time for comparing two long sequences, and

Species	Chromosome	Accession number	Length
<i>Homo sapiens</i>	chromosome X	GenBank:NC_000023.9	154.91
<i>Pan troglodytes</i>	chromosome X	GenBank:NC_006491.3	156.85
<i>Macaca mulata</i>	chromosome X	GenBank:CM002997.1	148.78
<i>Mus musculus</i>	chromosome X	GenBank:NC_000086.7	171.03
<i>Rattus norvegicus</i>	chromosome X	GenBank:NC_005120.4	159.97
<i>Bos taurus breed Hereford</i>	chromosome X	GenBank:NC_007331.4	88.65
<i>Canis lupus familiaris breed Boxer</i>	chromosome X	GenBank:NC_006621.3	123.87

Table 5.1: Information of the used dataset to evaluate the performance of GECKO. From left to right: species name, chromosome of origin, GenBank accession number and length in Mbp.

in multiple genome comparisons where several independent pairwise comparisons are typically performed sequentially, attracted our interest to develop a parallel strategy resulting in the workflow presented in the next section.

With the aim of testing the above contained workflow in the devised cloud-based infrastructure, we performed a massive exercise consisting on the comparison of the chromosome X of several mammalian species. Detailed information about the species, accession numbers and sequence length is contained in Table 5.1. The results and analysis of the performed benchmarking to the GECKO software are contained in Section 5.3.2.

### Multiple genome comparison parallel workflow

The GECKO workflow represented the starting point of this second implemented workflow. After a careful study of the internal data dependencies of the GECKO modules, we noticed that most of them were subject to an easy and efficient parallelization. As a result a two-level parallel approach to accelerate multiple genome comparisons was proposed. The first level is aimed at parallelizing each independent pairwise genome comparison of a multiple comparison study to a different core. This level is application-independent, we are using GECKO but any other equivalent software can be used. The second level consists on the internal parallelization of GECKO modules with evident enhancements in performance while results remain invariant.

In the first parallelization level, a master-slave tasks distribution approach to perform each pairwise genome comparison of a multiple genome comparison study in a different core (see Figure 5.5.A) has been applied. The second level follows a master-slave approach as well, where the slaves calculate the partial result of the modules composing GECKO (see Figure 5.5.B). This approach considers at

both levels as many slave processes as cores being used. The master process reads the set of tasks from a workload file, which is generated by a previous mapping process. Later the master distributes the tasks, assigning the cores more tasks as soon as they become idle.

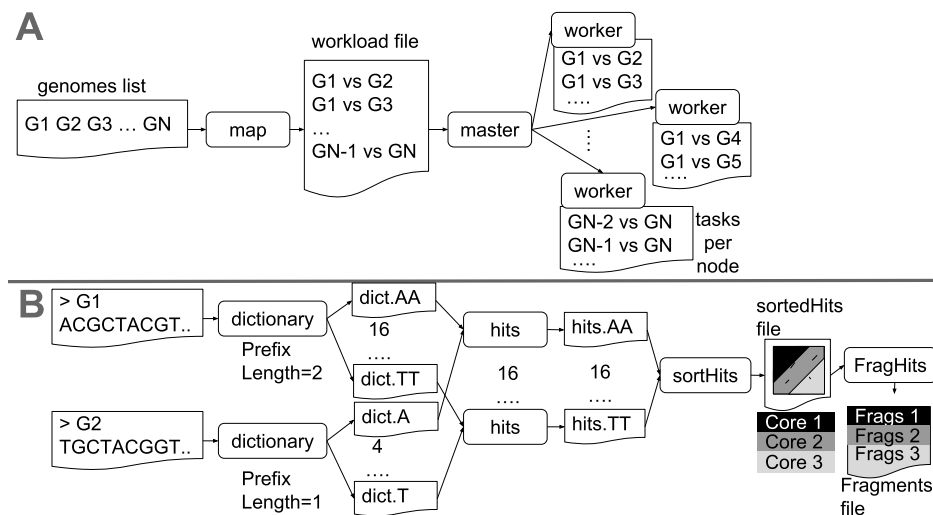


Figure 5.5: The parallelization levels applied to the multiple genome comparison workflow. Sub-figure A outlines how the strategy starting from a list of genomes ends performing each independent pairwise genome comparison in a separate worker. Sub-figure B shows the performed parallelization within the internal modules of GECKO.

Considering the high number of I/O operations performed by GECKO modules, the master is assigning more than one task per core in order to overlap I/O and computation. Additionally, this is done to reduce the overhead introduced by sending tasks in separated messages. Depending on the number of cores and the selected prefix size, the *tasks\_per\_core* value is either 2 (for number of cores power of 2) or 4 (for number of cores power of 4) in order to always have more than 1 task per core. For example, if we use 2 cores and a prefix size of 1, the *A* and *C* letters will be assigned to one core and the *G* and *T* to the other, resulting in 2 tasks per core.



### Genome-Wide Association Study workflow

The third real-world application consists of a Genome-Wide Association Study pipeline/workflow, which is not an individual development of this thesis as the previous two real-world workflows, but a participation in a collaborative work with multidisciplinary research groups. The workflow is composed of a set of interconnected tools (an overview of the workflow is given in Figure 5.6). Before starting the computation, all the CEL files (i.e. input files) to be analysed should be uploaded to the file system. A CEL file contains the raw data generated from a SNPs microarray analysis for a single patient containing details of its genotype. The first tool being executed is the birdseed algorithm [51], which is implemented in the Affymetrix Power Tools package. This tool produces a table of genotype calls for each of the different probes/SNPs on the microarray. A second tool filters the previous table removing SNPs that do not vary across different patients. Using the filtered table, a third tool converts the birdseed algorithm output, which is Affymetrix specific, into the standard Variant Call Format (VCF) file. This last tool is called once per CEL file, therefore it represents an embarrassingly parallel problem suitable for data-parallel execution. The system has been tested using as input data 8 CEL files of 66 MB each. The result of such execution is presented in Section 5.3.2.

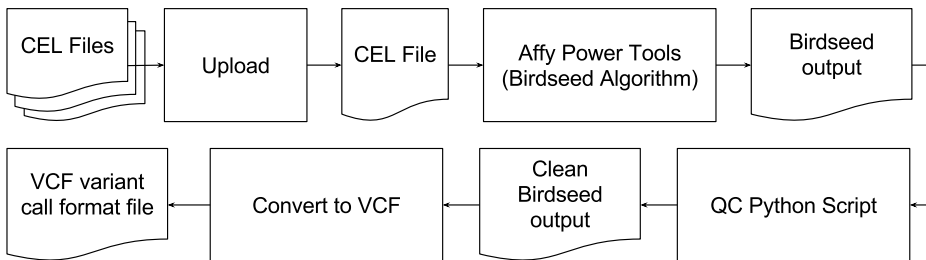


Figure 5.6: The GWAS workflow.

## 5.2. Evaluation metrics

The devised scheduling mechanism was designed considering four performance criteria: jobs queued time, makespan, throughput, and resources utilisation. The aim is minimising the average jobs queued time and makespan, and maximising the system throughput and resources utilisation (in the mentioned order). It is important to note that there exists a trade-off between the metrics. For instance,

if the average jobs queued time is reduced but there is a long-running task, which is queued for a long time, the makespan could be affected. This and other affecting factors are discussed in next sections. With some more detail, the definition of each of the metrics is the following:

- **Queued time:** this criterion represents the time a job –ready to run– has been waiting to be executed. It indicates the effectiveness of the system with regards to the sharing of the computing resources. In this work, we are evaluating the average and also the standard deviation. It is important to note that with a FCFS algorithm if the longer jobs are concentrated at the beginning the average queued time increases rapidly. With the devised algorithm the average queued time is expected to be reduced, long jobs are a little bit postponed but they never enter into starvation since their priority is dynamically increased with the queued time.
- **Makespan:** a metric calculated as the time since the first task of the workflow enters the batch system until the last task is reported to be finished.
- **Throughput:** this metric represents the number of jobs finishing per time unit. It is calculated as the number of jobs of the workload divided by the makespan.
- **Resources utilisation:** this metric indicates the average percentage of the system that is used during the workload execution. This time is calculated as the sum of the execution time of all the tasks divided by the makespan and the available number of nodes. It indicates if the scheduling is efficient enough in terms of taking profit of the instantiated machines, what is particularly important in pay-per-use environments such as cloud computing is.

### 5.3. Experiment results

In this section we evaluate first the different mentioned evaluation metrics (see Section 5.2) using the configuration parameters of both the job priority and node allocation policy of TORQUE (as described in Section 4.1.2) in Section 5.3.1. The results of each of the metrics have been separated into different subsections. Secondly, the overall system behaviour using an auto-scaled cluster following the strategy defined in Section 4.2 is evaluated in Section 5.3.2 using the 3 previously explained real-world workflows.

### 5.3.1. Results of the performance metrics

A small testing static infrastructure composed of two computing nodes has been used. The rationale is to reduce the external influence to the scheduling decision of several factors such as the auto-scaling strategy. Thus, the base FCFS scheduling algorithm and the one devised in this thesis can be compared in a controlled environment. Next subsections, ordered by how important they have been for the devised scheduling strategy, show the evaluation results.

#### Queued time

Figure 5.7 illustrates the average queued time of the jobs composing the different workloads together with the standard deviation. The results obtained with the priority-based scheduler outperform the ones of the FCFS scheduler in all the tested workflows, having both lower average and standard deviation values. This proves that the values of the configuration parameters for the priority calculation were correctly selected, considering the queued time as the first metric to be optimised (minimised in this case). Nevertheless, taking a closer look at the mentioned figure, we can observe that differences between FCFS and the priority-based scheduler differs from one workflow to another. Next paragraphs discuss the possible reasons of these differences.

On the one hand, the sets of independent tasks of the workloads *test-1*, *test-2* and *test-3* are composed of more small tasks than long-running tasks. In addition, the majority of the small tasks represent the final tasks of the workload. Given this two patterns of the mentioned workloads, it is expected that the FCFS algorithm will behave worse because of the high number of small tasks of the end. These tasks will have a longer waiting time in the queue proportional to the execution time of the long-running tasks present at the beginning of the workload. In contrast, the devised priority-based scheduler will assign a higher priority to the small tasks of the end, therefore they will execute before the long-running tasks thus reducing the average queued time.

On the other hand, the workloads of the 3 synthetic workflows *Cybershake*, *Ligo* and *Montage* have internal data dependencies between the tasks. These dependencies reduce the number of waiting tasks in the queue ready for execution, therefore the effect of the scheduling strategy is lower compared to sets of independent tasks. In scheduling cycles with a big number of tasks waiting in the queue, the distribution of tasks duration plays a more important role (Figure 5.15 in Section 5.4.1 contains the tasks duration distribution):

1. The more homogeneous the tasks duration are, the more similar the performance of the FCFS and the priority-based schedulers is (this is the case of the *Ligo* and *Montage* workflows). Although at first glance the tasks duration distribution of the *Ligo* workflow could look like heterogeneous, it is homogeneous in its period with more independent tasks, therefore the scenario is similar to the *Montage* workflow.
2. The queued time is lower in the priority-based scheduler when the tasks duration distribution is not homogeneous and there are more small tasks than long-running tasks without dependencies (as it can be observed in the tasks duration distribution of the *Cybershake* workflow).

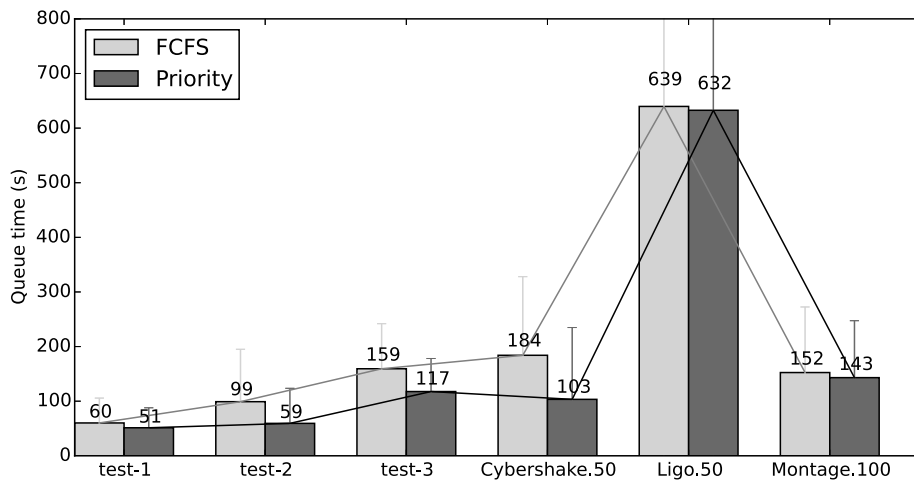


Figure 5.7: Average queued time of the different workflows scheduled by the two compared scheduling algorithms. The ‘light grey’ bars represent the time for the FCFS algorithm, whereas the ‘dark grey’ bars represent the time for the priority-based scheduler. The ‘error bars’ indicate the standard deviation of the queued time (in the Ligo.50 bars the standard deviation is not represented to limit the size of the Y axis, the error bars of such case reach approximately 1500 seconds). Lines connecting the bar values have been included to make it easier to interpret it.

## Makespan

The makespan of the different workloads is represented in Figure 5.8. As it can be observed the makespan has been reduced with the devised scheduling strategy in 3 out of the 6 workflows (i.e. *test-3*, *Ligo* and *Montage*). However, it has been increased in the other 3 cases. These are the expected results, because we aimed at pointing out the advantages and limitations of the designed scheduling mechanism. Next paragraphs provide the explanation of the obtained results.

In order to determine the reasons why some workflows reported a higher makespan, they are going to be inspected in more detail considering the included tasks and its duration (please refer to Section 5.4.1 for tasks duration). The first workload with higher makespan (*test-1*) is composed of 3 long-running tasks at the beginning, which in the case of FCFS are scheduled to be run at the beginning. Since the test infrastructure has 2 execution nodes, the first 2 long-running tasks enter the system for execution. After these 2 tasks finish their execution, the third one starts executing in one of the nodes. The other node executes the rest of small tasks present at the end of the workflow. In contrast, in the devised priority-based scheduling strategy first the small jobs are executed, therefore reducing their queued time, whilst the execution of the long running jobs is postponed. This is not necessarily bad if the number of long-running jobs is balanced with regards to the number of available computing nodes, but this is not the case of the mentioned workload. In the mentioned workload, the long-running jobs are unbalanced, thus reporting worse performance. A schematic representation of the previous discussion can be observed in Figure 5.9.

Similarly, the set of tasks *test-2* was designed to reinforce the previous analysis. In this case, more small tasks were included, but still keeping the unbalanced set of long-running tasks. The result is a similar situation as reported for *test-1* (i.e. higher makespan caused by the last long-running tasks executed at the end).

The scenario for the *Cybershake* workflow obtained from the workflow generator gallery is different. In this case, there exist data dependencies between tasks, therefore the priority value is only considered for tasks having all their dependencies satisfied. In the case of this workflow, there are only 2 long-running tasks without dependencies at the beginning (as illustrated in Figure 5.3). The difference in execution time of this 2 tasks unbalances the start of part of the tasks of the second level, which depend on them. This unbalance compromises the system performance with regards to the makespan, in particular for the devised priority-based scheduler, which postpones the long-running tasks.

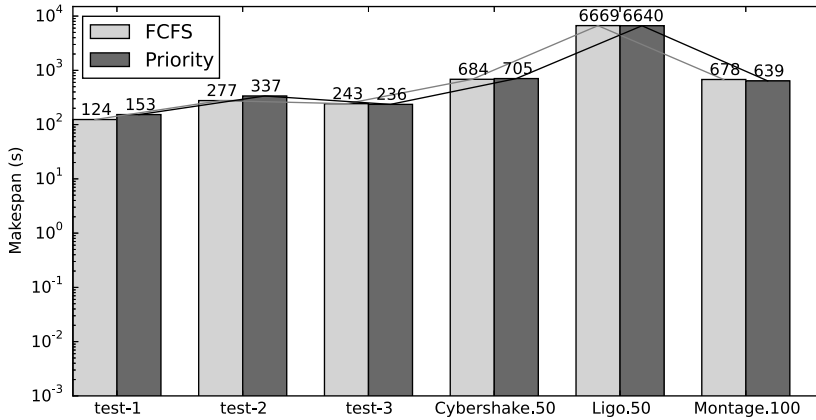


Figure 5.8: Makespan of the different workflows scheduled by the two compared scheduling algorithms. The ‘light grey’ bars represent the time for the FCFS algorithm, whereas the ‘dark grey’ bars represent the time for the priority-based scheduler. The Y axis is in logarithmic scale and lines connecting the bar values have been included to make it easier to interpret it.

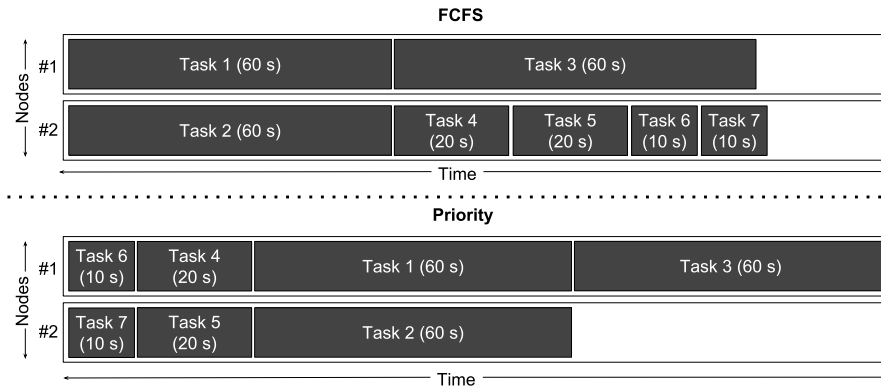


Figure 5.9: Performance of the scheduling algorithms for the unbalanced set of long-running tasks of test-1. The plot is divided into two parts. The upper half of the plot represents the behaviour of the FCFS algorithm, whilst the bottom half of the plot shows the performance of the priority-based scheduling algorithm designed in this thesis. It can be observed how the higher priority of the small tasks together with the unbalanced set of long-running tasks make the priority-based scheduler performs worse compared to the FCFS scheduler.

Throughput

Figure 5.10 contains the system throughput for the different synthetic workflows considering only one workflow at a time being submitted to the system. It is important to note that throughput value is inversely proportional to the makespan value, and since the number of tasks for each workflow is fixed, it would depend only on the makespan. That said, it is obvious to notice that the throughput value would be higher (i.e. better) for smaller values of makespan. This was the case in 3 out of the 6 workflows as reported in the previous section. The reasons of the different makespan values contained in the previously mentioned section are also valid for the different reported throughput values.

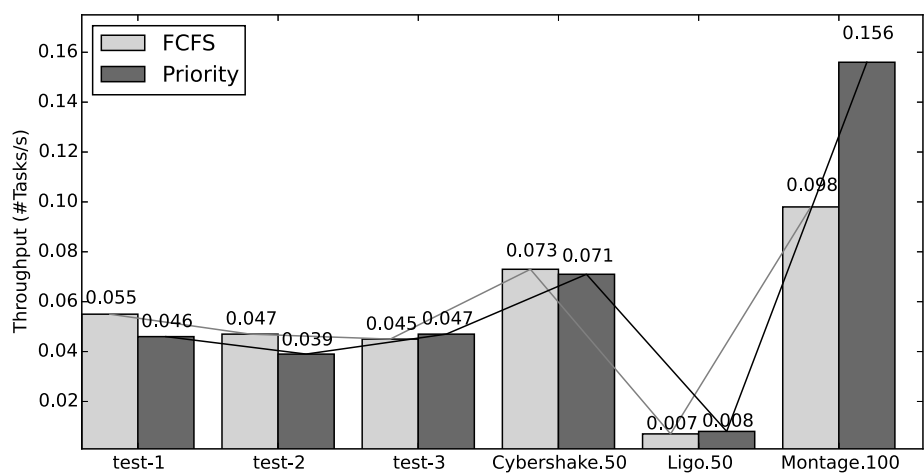


Figure 5.10: Throughput of the different workflows scheduled by the two compared scheduling algorithms. The ‘light grey’ bars represent the time for the FCFS algorithm, whereas the ‘dark grey’ bars represent the time for the priority-based scheduler. Lines connecting the bar values have been included to make it easier to interpret it.

Resource utilisation

The percentage of resource utilisation for each of the workflows using the two different scheduling strategies is represented in Figure 5.11. Similarly to the throughput value, the resources utilisation also depends on the makespan value. In this case, the resources utilisation is directly proportional to the total

execution time of the tasks (which is practically the same in different executions), and inversely proportional to the makespan value. Consequently, the resources utilisation value is better for the priority-based scheduler in 3 workflows. The reasons of the different makespan values are again valid to explain the resources utilisation values.

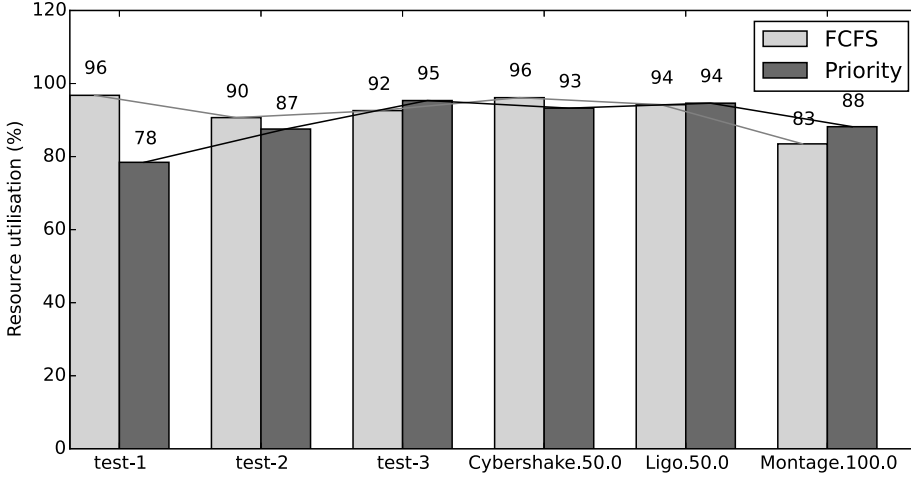


Figure 5.11: Resources utilisation of the different workflows scheduled by the two compared scheduling algorithms. The ‘light grey’ bars represent the time for the FCFS algorithm, whereas the ‘dark grey’ bars represent the time for the priority-based scheduler. Lines connecting the bar values have been included to make it easier to interpret it.

### 5.3.2. System behaviour using the auto-scaling strategy

This section is aimed at studying the system behaviour while using the developed auto-scaling strategy described in Section 4.2, whilst the previous performance evaluation sections were using a fixed-size execution cluster. An additional difference is that in this section the evaluation metrics are not considered anymore, instead the time reduction and application speedup are evaluated. Previous sections highlighted the performance of the priority-based scheduler of this thesis compared to the FCFS, both in synthetic and real-world workflows. In this section the aim is at evaluating the auto-scaling strategy, thus only the priority-based scheduler is used to remove the influence of having different schedulers. Next subsections show the evaluation results for the sequential and parallel versions



of the GECKO software for multiple genome comparisons, and for the GWAS workflow.

### **Multiple genome comparison sequential workflow**

This section reports the performance of the system while executing a multiple genome comparison study using the sequential version of the GECKO software and the mammalian sequences dataset described in Section 5.1.2. This dataset generates 21 pairwise genome comparisons following what is known as an all-versus-all study.

The overall sequential (1 core) execution time of the all-versus-all multiple genome comparison of the mentioned dataset is of 33.83 hours. But this time includes calculating the dictionary of input sequences for each comparison, what is not required as stated previously in Section 5.1.2. Therefore, calculating the dictionary just once for each sequence, the execution time is reduced to 16.5 hours.

Nevertheless, since each pairwise comparison inside the multiple genome comparison is independent, they can be executed in parallel in separate cores. The 21 tasks or pairwise genome comparisons were submitted to an auto-scaled cluster of up to 10 nodes. The auto-scaling strategy managed to up-scale the system thus reducing the execution time to a total of 2.33 hours with an efficiency of 70.82% compared to the optimal theoretical speedup (an acceptable speedup given the small number of tasks). After finalising the execution, the cluster was again down-scaled to the minimum number of instances (i.e. just the static instances).

### **Multiple genome comparison parallel workflow**

The previous section reported the experienced time reduction, due to the auto-scaling strategy, while executing a multiple genome comparison using the sequential original version of GECKO. As already mentioned, after studying the parallelization possibilities of GECKO, it was observed that some internal modules were subject to be parallelized in an efficient way. These modules include the sequences dictionary calculation, the determination of hits, the sorting of such hits and the calculation of the final alignment. All of them report a significant amount of execution time to benefit from a parallel strategy given the possibility of splitting the input data in smaller chunks. Therefore, such modules were adapted to run in parallel in the kind of infrastructures as the presented in this work. In this case, the execution cluster was auto-scaled before executing the

experiments in order to measure separately the benefits of the devised parallel strategy.

In this use case two different types of datasets were used. The first one composed of two sets of 30 and 40 bacterial sequences respectively to be used for testing the first parallelization level described in Section 5.1.2. The aim of this two sets is to generate a sufficient number of tasks in order to have a good application speedup. It is important to note that an all-versus-all study generates  $n * (n - 1)/2$  tasks, what accounts for 435 tasks in the case of the 30 sequences set, and for 780 tasks in the 40 sequences set. Figure 5.12 shows the application speedup for the mentioned set of sequences. The second dataset is composed of sequences with notorious differences in size, ranging from approximately 5Mbp to 420Mbp. The aim is to benchmark the parallel strategy in the experimental condition of heterogeneous workloads.

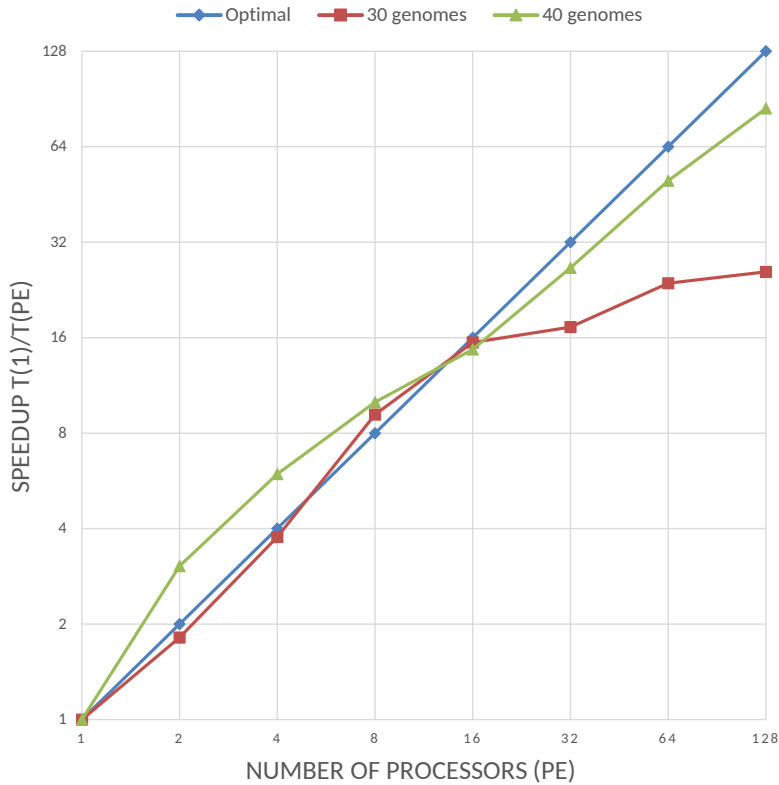


Figure 5.12: Speedup of the first parallelization level of the multiple genome comparison workflow. The X-axis represents the number of used cores, whereas the Y-axis shows the application speedup calculated as the sequential execution time with one core divided by the running time for the given number of cores.

The speedup of the first parallelization level (see Figure 5.12) suggests that the application is scalable. The reported speedup is good until 16 cores having a speedup close to the theoretical one or even superlinear because of the overlap between I/O and computation while executing several comparisons at the same time. From this number of cores onwards, the number of tasks is not sufficient in the 30 genomes series, thus its speedup starts to degrade. This is not the case of the 40 genomes series, which generates a higher number of tasks.

Figure 5.13 shows the speedup for the different modules composing the GECKO software. It is worth mentioning the effect of the size of the input sequences be-

fore analysing the speedup. The length of the input sequences is one of the main factors affecting the execution time of GECKO. Additionally, other components influencing the execution time are the similarity shared by the input sequences and the parameters selection, particularly the word or K-mer size chosen to calculate the hits or seed points from where to start the alignment (see Section 5.1.2 for more details).

Two interesting aspects can be observed in the sub-figures from A to D of Figure 5.13. First, the super-linear speedup achieved in some cases. This is caused by the fact of having simultaneous executions on each node overlapping computation and I/O operations. The second aspect is the reduction of the speedup. There are two possible reasons to explain this behaviour. The first reason is that particularly for the shortest sequence (i.e. E.coli series) the amount of work to be performed is not big enough to take profit of a parallel strategy. The second reason is the amount of I/O load the application has. The computation itself is parallelized but not the I/O, therefore all the time spent reading/writing the big input/output files seriously reduces the obtained speedup. A more detailed discussion of the speedup can be found at [85].

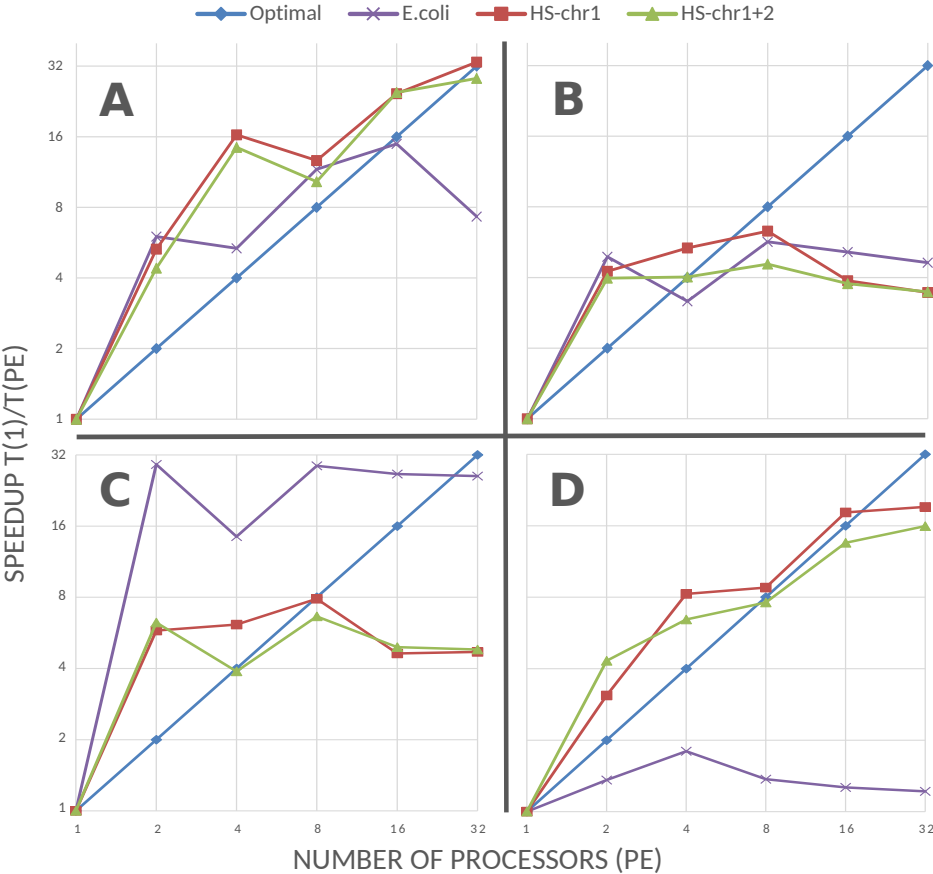


Figure 5.13: Speedup of the modules composing the GECKO workflow. A: Dictionary step speedup; B: Hits step speedup; C: Sort hits step speedup; D: FragHits step speedup. The X-axis represents the number of used cores, whereas the Y-axis shows the application speedup calculated as the execution time in sequential fashion using one core divided by the running time for the given number of cores.

An attractive point that can be extracted from the obtained speedups is that since the different modules report good efficiency levels until different number of cores, this means that the workflow is particularly suitable for dynamically scalable environments such as the presented cloud computing infrastructure. Extracting the best number of cores for each of the modules, during task submission such values can be used to improve the efficiency and to reduce the execution cost.

### Genome-Wide Association Study workflow

The Genome-Wide Association Study workflow was executed via the Galaxy workflow management system connected to the auto-scaled cloud-based execution cluster designed in this thesis. The input dataset is composed of 8 input files of 66MB each, which generate 8 independent tasks. For the sequential part of the workflow consisting of the birdseed algorithm and data filtering, a longer runtime compared to the execution time of the original publication [40] was obtained. The cause resides in that these tasks were mapped to the node hosting the Galaxy web server with the consequent overlap and influence of the Galaxy processes to the execution time of the tasks. However, in the VCF conversion step a significant difference to the original publication can be seen. This step is memory-bounded, therefore the ratio of CPU cores (or simultaneous tasks) to GB of RAM has a large performance impact.

This can be observed comparing the mean runtime of the original publication, which was of 4970 seconds in the same cloud environment of this experiment, to the execution time in the different machines of the auto-scaled cluster. First, the task scheduled to the Galaxy main node reported a mean execution runtime for a single VCF conversion of 1190 seconds. Second, the 4 VCF conversions executed in parallel by the permanent compute node, resulted in an execution time of 1416 second on average per conversion and a memory consumption of 2 GB each. Third, the rest of the conversions (3 in this case) were scheduled in a dynamically started VM with 8 vCPUs and 8 GB of RAM reporting a mean runtime of 569 seconds in this machine. In summary, all the conversions took an average of 1756 seconds including the overhead introduced by the Galaxy workflows management system. The 4970 seconds of the original implementation have been reduced to 1756 seconds using 3 nodes, what gives an efficiency value of 94.34%. These times exclude the time required to upload the data to the cloud environment, which in any case is not too big for approximately 500MB.

## 5.4. Main factors affecting the scheduling and auto-scaling mechanisms

In this section, the main factors affecting the system behaviour are discussed in different subsections:

- First, the influence of the **task duration distribution**, particularly the point in time when the long-running tasks are submitted and the balance

between the number of tasks and computing nodes are discussed.

- Second, the next section discusses how the system behaves when the **accuracy of the tasks' walltime** or runtime estimates is not good. In this second part, the same configuration parameters as in the first evaluation will be used.
- To finalise this section, the alteration produced by the **provisioning delays of VMs** within the cloud environment is discussed.

#### 5.4.1. Task duration distribution

One of the main factors affecting the system behaviour is the distribution of the tasks duration of the faced workload. First, the combination of short and long-running tasks is important. In workloads including a lot of long-running tasks, the average queued time of the short ones could be affected. But not only the number of each of the tasks groups with regards to duration as shown in the histograms of Figure 5.14 is important. Second, it is also equally important when each of task types are submitted as illustrated in Figure 5.15. This can influence the performance of schedulers depending on the tasks submission order such as the FCFS algorithm.

For instance, if the tasks distribution concentrates most of the long-running tasks at the beginning, the average time that a task spends in the queue will be increased (this is the case of the workloads *test-1* and *test-3* as reported in Section 5.3.1). Additionally, if the number of long-running tasks is not balanced to the number of available compute nodes, the makespan could be increased and thus the throughput and resources utilisation efficiency reduced (as reported for the workloads *test-1* and *test-2* in Section 5.3.1).

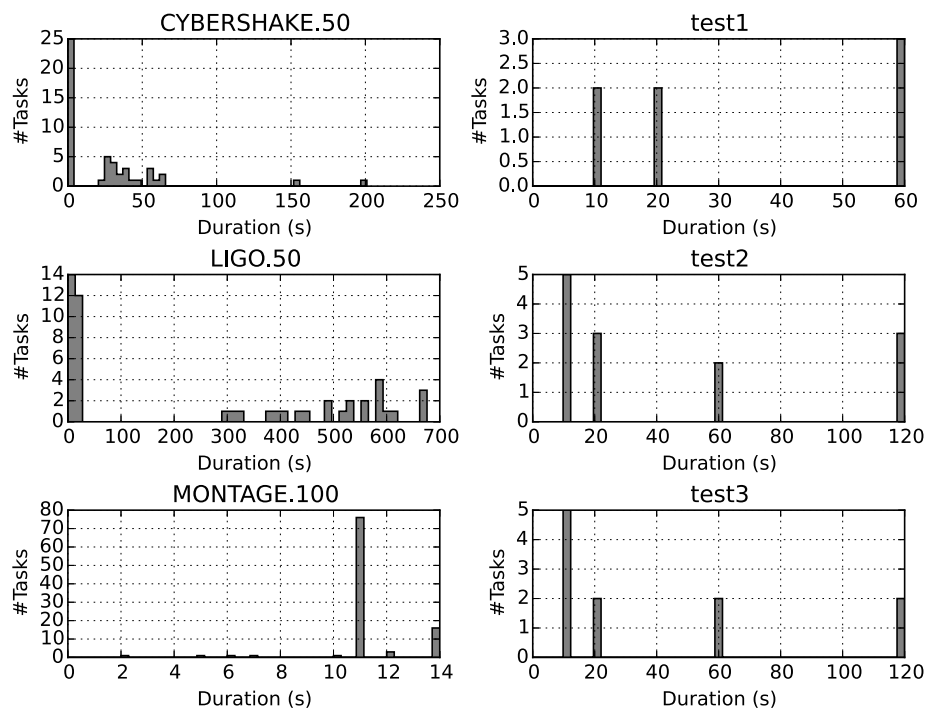


Figure 5.14: Histogram of tasks duration for the different workflows. The X-axis represents the tasks duration from 0 till the longest running time. The Y-axis shows the number of tasks of the given duration.



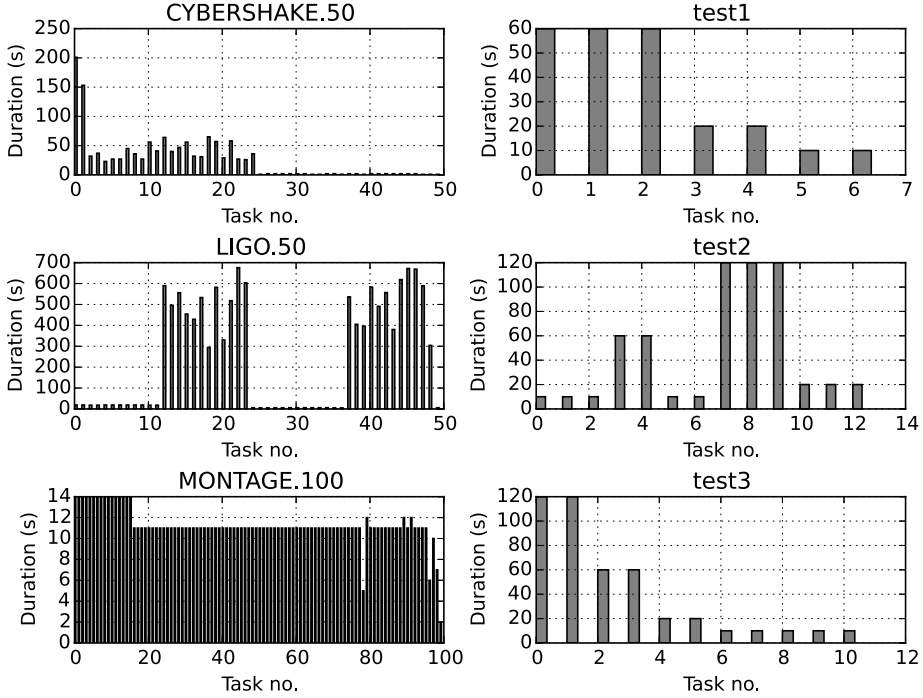


Figure 5.15: Barchart of tasks duration for the different workflows. In this case the tasks are arranged in the X-axis by submission order. The Y-axis accounts the specific task duration of the given task.

In [58] the authors already noticed that for the synthetic workflows described in previous sections, especially for Montage and CyberShake, the short runtimes of their tasks made their dynamic algorithms perform better relative to their static one. They demonstrated this assumption by adjusting the runtime of each task multiplying it by a fixed scaling factor.

#### 5.4.2. Inaccuracies in the task runtime estimates

Another important component affecting the scheduling performance is the accuracy of the tasks' walltime or runtime estimates provided by the user. In order to assess how this fact affects the scheduling decisions performed by the scheduler strategy, we are following the same evaluation approach to measure again all the evaluation metrics.

It is important to note that the job priority and therefore the job scheduling performed by the priority-based scheduler relies on estimates of tasks runtimes. Typically, the more accurate the task runtime is, the better the scheduling and provisioning decisions are. In [58] the authors claim that such assumption is often reasonable, since it is possible to obtain workflow performance characteristics from preliminary runs [12, 24, 47]. Some workflow management systems, such as Galaxy [13] and Swift [100], include data provenance techniques, which gather task runtime estimates among other values related to the workload. Workflow descriptions may in turn be *a priori* annotated with these estimates, and *a posteriori* with the real execution time. In practice, however, these estimates can not be easily extrapolated to other input data, particularly when input parameters have strong effect in the runtime. Consequently, in this section we examine how these inaccuracies affect, in a parametric study that varies the execution time from the real runtimes known in advance to this time plus an extra time following a random distribution from 0 to the task duration. Underestimates are not evaluated in this study since in such a situation TORQUE would remove the job from execution in the current configuration. However, this behaviour can be easily changed to be more permissive, allowing a certain percentage of inaccuracy per job.

Figures 5.16 and 5.17 show the tasks duration histogram and bar chart respectively including the random extra time added to the original walltime. Next paragraph discusses how these changes in the runtimes affect the average tasks queued time and makespan of the scheduling strategies for the workloads evaluated in Section 5.3.1. The rest of the evaluation criteria (i.e. throughput and resources utilisation) are not included in this section because they are proportional to the makespan.

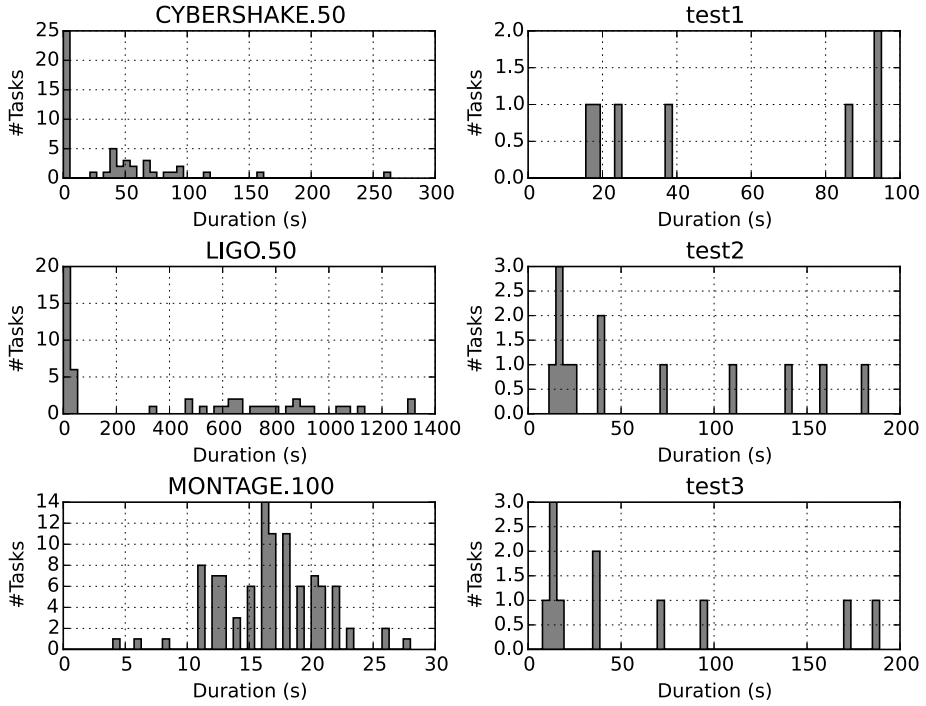


Figure 5.16: Histogram of tasks duration for the different workflows including a variable extra random time. The X-axis represents the tasks duration from 0 till the longest running time. The Y-axis shows the number of tasks of the given duration.

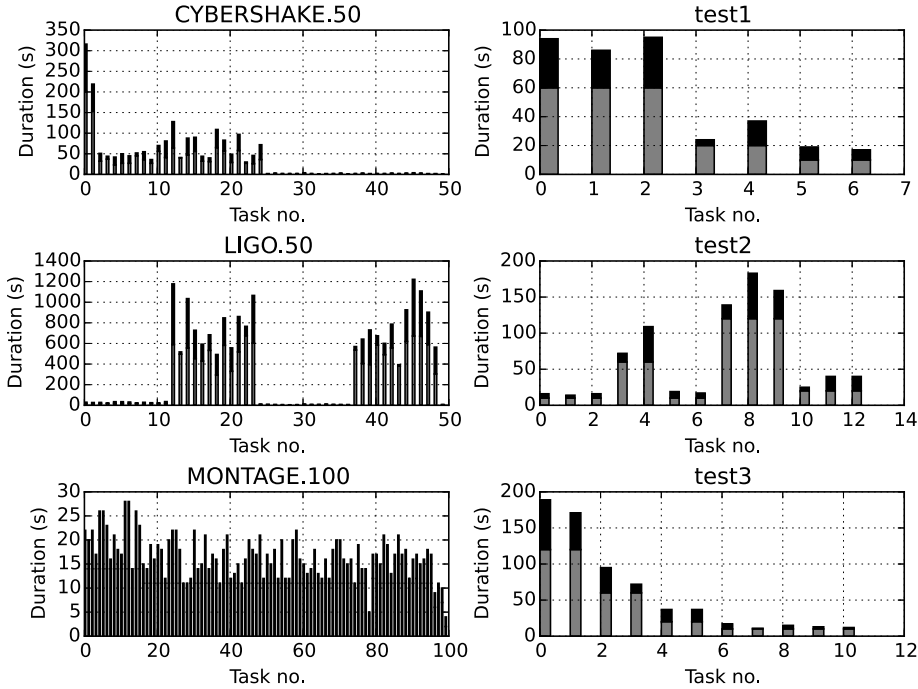


Figure 5.17: Barchart of tasks duration for the different workflows including a random extra time. In this case the tasks are arranged in the X-axis by submission order. The Y-axis accounts the specific task duration of the given task. The grey part of the bar represents the original time, whilst the black part accounts for the extra random time added to the original walltime.

The average tasks queued time of the different workloads with the modified walltime can be seen in Figure 5.18. The queued time for the priority-based scheduler was shorter in all the different workloads compared to the FCFS algorithm in the original benchmark. Therefore, Figure 5.18 only displays the queued time of the priority based scheduler for the original workloads compared to the modified one including the walltime inaccuracies. As it can be observed, surprisingly except for one workload (*test-2*), the rest of them report on average either a similar or shorter waiting time in the queue. The *test-2* workload changed its task duration distribution significantly from having more small tasks than long-running tasks, to having a small number of short tasks and a big amount of medium and long-running tasks. This made the priority-based scheduler to select the short tasks without any doubt, but this was not the case for the bigger ones

thus increasing the average waiting time in the queue. For workloads with similar queued times (*test-1*, *test-3*, *Cybershake.50* and *Ligo.50*), the tasks duration distribution has not significantly changed therefore the scheduler makes the same decisions. However, for the *Montage* workflow, whose task duration distribution was similar along the different task sizes, modifying the walltime has contributed to help the scheduler performing a better decision, reducing the average queued time, in the situations where the runtime estimates were pretty similar.

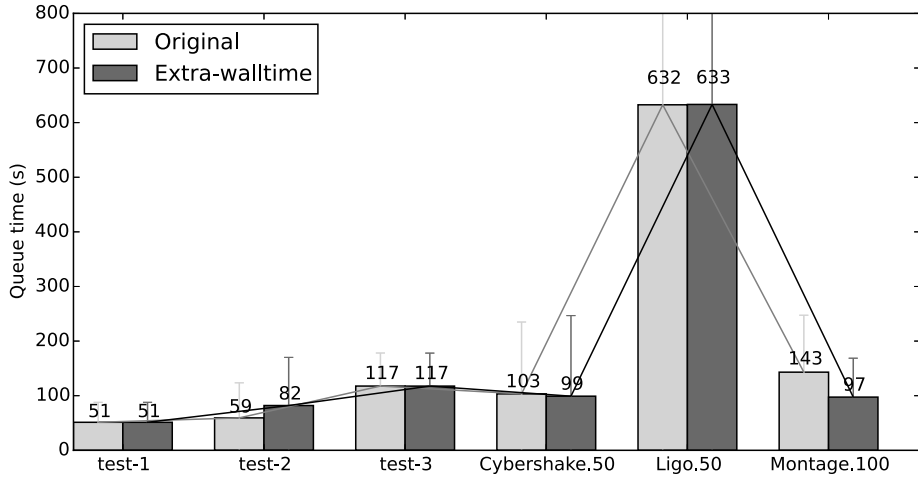


Figure 5.18: Average queued time of the different workflows scheduled by the two compared scheduling algorithms including a random extra time. The ‘light grey’ bars represent the time for the priority-based scheduler with the original walltimes, whereas the ‘dark grey’ bars represent the time for the same scheduler with the inaccurate walltime. Lines connecting the bar values have been included to make it easier to interpret it.

With regards to the makespan metric, Figure 5.19 shows the comparison of the makespan for the different workflows using the priority-based scheduler with the original (accurate) and the inaccurate walltimes. As it can be observed, for most of the workflows the makespan value has not significantly changed. In 4 workflows it has been slightly increased. In these cases the execution order of some short tasks scheduled to be executed at the beginning was shifted to later periods thus unbalancing the workload in the compute nodes (this scenario is illustrated in the *Unfavourable* part of Figure 5.20). This is happening because the priority value of such tasks was very similar to the one of other medium or long-running tasks

with the original walltime, therefore adding some extra time changed the priority and in consequence the order. In contrast, the makespan has been reduced for 2 workloads. In these scenarios, the extra walltime assigned to the short running tasks moved their execution to the end, what fills the ‘idle’ gaps left by the long-running tasks at the computing nodes (a similar situation is represented in the *Favourable* part of Figure 5.20). The behaviour for the throughput and resources utilisation metrics can be extrapolated from this case, since they are inversely and directly proportional to this criterion respectively.

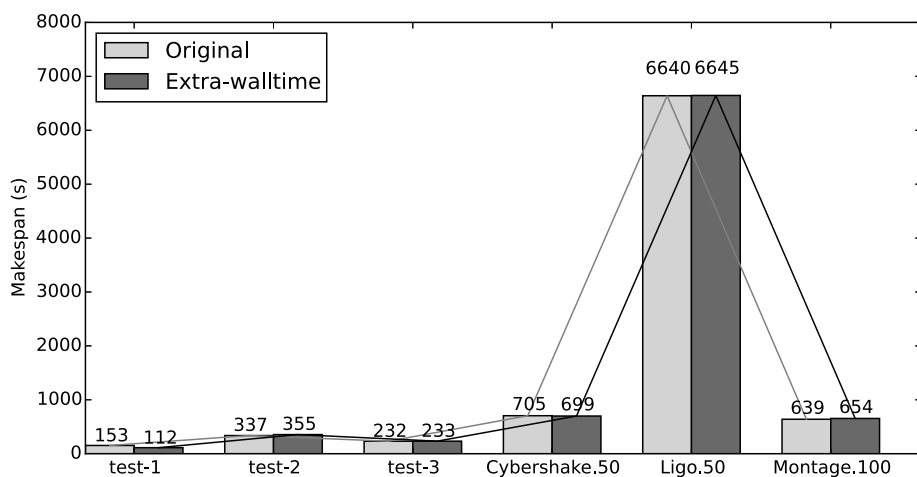


Figure 5.19: Makespan of the different workflows scheduled by the two compared scheduling algorithms including a random extra time. The ‘light grey’ bars represent the time for the priority-based scheduler with the original walltimes, whereas the ‘dark grey’ bars represent the time for the same scheduler with the inaccurate walltime. In the Ligo.50 bars the standard deviation is not represented to limit the size of the Y axis. Lines connecting the bar values have been included to make it easier to interpret it.

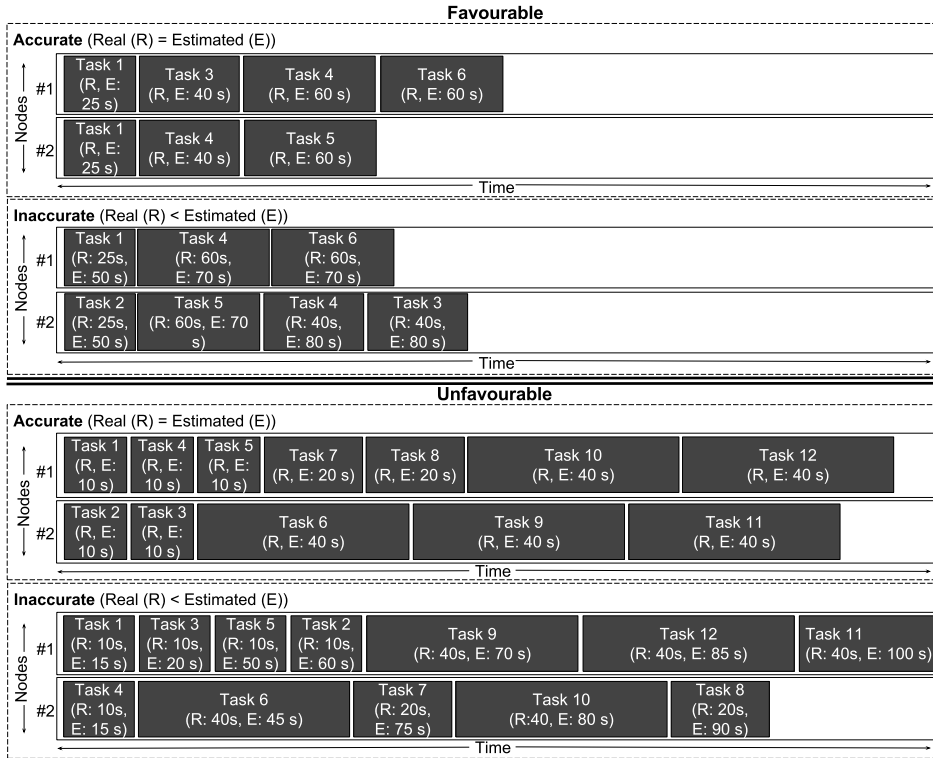


Figure 5.20: Influence of the inaccuracies in the tasks runtime estimates to the makespan metric. The figure is split into two parts (upper and bottom). The upper part represents a favourable situation where inaccurate runtime estimates reduce the makespan. The bottom part outlines an unfavourable case where the makespan has been increased. For each of the parts, two scenarios have been represented: accurate and inaccurate runtime estimates. It is important to note that the priority value of the tasks depends on the runtime estimates (E) and not in the actual execution time (R).

### 5.4.3. Provisioning delays

The delay of cloud resources provisioning is one additional point to consider in the process of making the decision of either adding or deleting resources. This time refers to the period between a resource is requested until it is finally available. Typically most of the public cloud providers bill the resources from the instant they are requested by the user until they are released, including the provisioning

and releasing delays in the cost the user should cover.

The provisioning delay depends, to a greater or lesser extent, on multiple factors. Usually the considered factors (as studied in [60]) are the cloud provider, the time of the day the resources are requested, the VM operating system image size, the VM instance type, the availability zone, the number of simultaneous requested resources and if they are spot instances or not (spot instances enable users to bid on unused instances to reduce costs. The price of such instances fluctuates based on their supply and demand. The big variability of the provisioning delay has made cloud simulators such as CloudSim and adaptive resources provisioning models such as [44, 55] to consider it.

In [58] the authors reported that in cases of delays of more than one minute, the incurred cost of a significant number of their simulations was increased sometimes by up to a factor of 2, exceeding the budget constraints they are considering. In contrast, the cost of their dynamic algorithms vary, but never exceeding the imposed cost constraints. Considering this important issue, we are dynamically studying when it compensates to include a new instance to the execution cluster. Briefly, the devised auto-scaling strategy adds a new instance to the cluster when the runtime of waiting tasks in the queue is higher than the provisioning delay of the instance type required by the task of highest priority.

In order to estimate the provisioning delay of the different instance types available in the used OpenStack cloud infrastructure, we performed a simple study to calculate the average delay and its standard deviation. The study was performed requesting resources at different times of the day in order to avoid biases on the system faced workload. The results shown in Figure 5.21 follow a similar distribution of provisioning delay per flavour as reported for EC2 in [60].



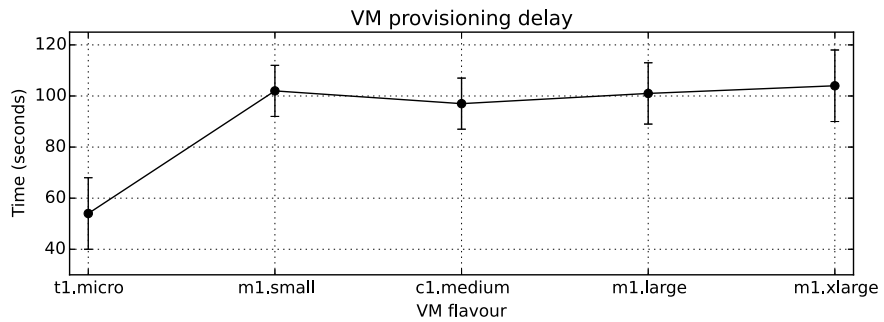


Figure 5.21: Provisioning delay of the different instance types of the used Open-Stack cloud infrastructure. The X-axis contains the different instance types, whereas the Y-axis reflects the average time in seconds to provision the instance.



# 6 Conclusions and Future Work

---

## HPC and cloud computing

In the Big Data era, the massive production of vast amounts of data in numerous fields such as astronomy, health, Smart Cities, the Internet of Things and social networks has shifted the bottleneck from data acquisition to data analysis and interpretation. Efficiently analysing the tremendous amount of data that we have available has turned out to be a very complex issue. High Performance Computing strategies represents an interesting alternative due to their proven success while dealing with large datasets. However, in general such techniques require the use of large and expensive computing infrastructures, which are typically shared between different groups. This produces some consequences such as the overload of resources (in number of tasks and their duration); the necessity of the batch execution mode; a big amount of users; a high I/O load due to the data volume; etc. Cloud computing with its scalability and pay-per-use properties represents an interesting alternative to build HPC infrastructures. Obviously, since cloud computing is not particularly targeting such purpose, some research is being performed to evaluate it as a different HPC infrastructure.

In the context of cloud computing used as a HPC infrastructure, we have first evaluated the cloud computing services models (i.e. IaaS, PaaS and SaaS). IaaS has been chosen as the base model due to its higher performance for HPC applications, and its wider number of configuration possibilities, which allow a more direct access to the underlying infrastructure. Secondly, we have identified the tasks to be performed in the cloud environment: data transfer, browsing of

available services/tools, navigation through the hierarchy of available services, services composition, execution of services, process scheduling, and resources auto-scaling. Either state-of-the-art open source software has been used or we have developed additional software to enable us evaluating our contributions for the mentioned tasks. Thirdly, a priority-based tasks scheduling strategy has been proposed optimising a number of different metrics. Fourthly, the use of the devised cloud-based infrastructure has been made easier by interconnecting it with existing software clients and workflows management systems, which exploit the underlying infrastructure. Finally, we have selected several use cases to benchmark the performance of the devised strategies.

### **Base software for the cloud computing environment**

After evaluating different cloud computing service models and different providers, the OpenStack middleware was selected as the base IaaS cloud computing solution. Several modules (most of them already available and some developed in this work) have been included to improve the authentication mechanisms (LDAP and DirGrid), data storage and transfers (Ceph file system, GridFTP, GlobusOnline and CloudFuse), computation (TORQUE, Maui and Dynamic TORQUE), and user-friendliness (jORCA, mORCA and Galaxy). The result is a seamlessly integrated infrastructure, which allows end-users performing their analyses on a cloud-based HPC infrastructure via user-friendly interfaces.

### **User authentication**

The used authentication mechanisms (LDAP and DirGrid) ensure the system is correctly secured, allowing only authenticated users to interact with it. Authentication is required in several system components. For instance, it is used to access the data storage and to perform data transfers, to access the cloud computing web-based management interface, and also to execute scientific analyses via the software clients. Having a centralised location for storing the information of user accounts and their corresponding permission simplifies on one hand the tasks of system administrators, which have a single point to modify user rights. On the other hand, from the point of view of the users, they just need to remember one key-pair of user and password for the different services provided.

## Tasks scheduling and resources auto-scaling

Regarding computation, we contribute to allow TORQUE working with a dynamical number of worker nodes allocated in an OpenStack cloud. It is really important to maintain a dynamic execution cluster since it is not efficient to maintain an underutilised big cluster, and neither we want our jobs to be waiting too much time when new workers to execute them can be dynamically instantiated. With the workflow use cases used in this work, we have demonstrated that the auto-scaling feature of cloud computing would help extracting a good efficiency in applications with a varying demand on the number of resources during execution. The importance of such point can be observed in the amount of research carried out using different distributed resources managers as stated in the related work section. The auto-scaling strategy removes the limitations of equivalent software (see Table 2.1), which mainly work with simulated environments.

Although Dynamic TORQUE conforms the base of the reported auto-scaling strategy, it has been modified to map the devised infrastructure. Besides, we are not using the monitoring part included in the original implementation. The main changes reside in the auto-scaling decision mechanism and also in the scheduling strategy as described in Chapter 4. The first improvement is that the amount of required resources specified during the job submission is considered, instantiating worker nodes with enough resources. This contrasts with the assumption that all the jobs require the same number of resources made by the original implementation. With the previous assumption, the submitted job could not fit the resources of the new worker, what could lead to inconsistencies. The worker would be initiated, but at the end it will not be used because TORQUE realises that the job it was created for, is actually not fitting there. An additional improvement is the way the idle workers are deleted. The original version of Dynamic TORQUE just considered the time a worker had been idle, deleting it immediately. Such behaviour has been modified, triggering the worker deletion at the same point as the original implementation, but not actually deleting until its billing period is reached. The billing period is a parameter provided by the cloud vendor, which can be specified in the configuration file of the auto-scaling strategy.

It is important to note the flexibility of the devised approach, made possible by the designed software architecture. First, it is possible to use different distributed resource managers by implementing parsers for the output of the commands to query the status of the execution nodes (e.g. *pbs\_nodes*), and to retrieve the list of jobs waiting in the queue (e.g. *qstat*). Second, its architecture allows to use other cloud solutions as well. In this case, just a new class inheriting the abstract class implementing the specific cloud API calls would be required. These API

calls are the ones for creating and deleting worker nodes, and for monitoring their status at the lowest abstraction level.

The separated deployment of the auto-scaling strategy running alongside TORQUE presents two major benefits regarding usability and stability. First, it is transparent to the end-users, which submit jobs to the TORQUE queue as if they were using a regular TORQUE execution cluster, without requiring specific cloud computing knowledge. This removes the need of learning a new job submission language, and of modifying already available software prepared for off-the-cloud TORQUE clusters. Second, if the Python scripts managing the auto-scaling crash, TORQUE can continue working without service interruption. Of course the cluster would not be automatically scaled until the operation of the auto-scaling strategy is restored, but just re-running the auto-scaling part, the system would recover from the last stable status.

We have contributed with a priority-based scheduler which optimises a number of different performance metrics. The chosen metrics in order of importance for our scheduling strategy have been: average jobs queued time, makespan, throughput and resources utilisation (the definition of these metrics are contained in Section 5.2). Our aim has been to reduce the average jobs queued time, and at the same time the makespan, whilst increasing the throughput or number of executed tasks per second, and the resources utilisation. The parameters to calculate the priority of a given job have been chosen considering the previously mentioned optimisation objectives for the performance metrics. In case the user would like to optimise different metrics, a simple re-configuration of factors producing the priority value would be enough. A backfill policy has been used to prevent an inefficient use of the available resources in the system in situations with a mixture of long-running tasks requesting a big amount of resources and short tasks consuming a small number of resources (see Section 4.1.2). This kind of policy usually increases resources utilisation and theoretically does not affect the starting time of long-running tasks as planned by the scheduler, since only jobs fitting backfill windows are selected. Regarding the node(s) allocation policy to select the nodes(s) where to execute a given job, TORQUE has been configured to prioritise static over dynamic nodes in a first level, and the nodes with the least amount possible of resources (in other words a best-fit approach) in the second level. The reasons of selecting this policy are that in a cloud environment, firstly all VMs have an associated cost, so it is better to prioritise static nodes over dynamic ones. Secondly, VMs with more resources usually have a higher cost. The mentioned policy would leave sooner idle the workers with the bigger amount of resources, which will be in turn deleted by the auto-scaling strategy.

## Accessing the cloud: clients

Traditionally, software clients and workflows management systems have simplified the exploitation of complex computing infrastructures via user-friendly graphical user interfaces. However, already existing software clients such as jORCA or mORCA, and workflows management systems such as Galaxy, were designed and developed before the appearance of cloud computing thus not considering its specific properties. *A priori* data uploading and the call-by-reference to analytic services were not included in the original functionality of such software clients. We contribute aiming to remove such limitations of the mentioned software clients. First, data management plugins have been developed for such software clients. These plugins use existing data transfer protocols (i.e. SSH File Transfer Protocol (SFTP) and GridFTP) and applications (i.e. GlobusOnline) to perform reliable transfers of the user data. Second, a RESTful Web Services front-end has been designed and implemented. Such front-end provides a uniform standardised representation of various operations including service invocation, monitoring and results retrieval. In addition, it allows executing services using references to the input data instead of invoking them with the actual input values, which could be potentially large files. Third, the Galaxy workflow management system has been also configured to work with the mentioned data transfer plugins and has been interconnected to the cloud-based auto-scaled execution cluster of this thesis as well. All this effort translates into a higher abstraction level and therefore an easier use of the underlying cloud-based HPC infrastructure by the end-users.

## Validation and benchmarking: use cases

The use cases selected to demonstrate the performance of the devised strategies are conformed by synthetic and real-world applications. Both, the synthetic workflows (used in related works), and the real-world applications (result of this work and derived from the collaboration with experts of the bioinformatics and biomedicine fields) represent a good benchmarking suite composed of CPU-intensive and I/O-bounded applications with regular and irregular computation patterns. In particular, the simulation of the synthetic workflows, which have typical computational patterns, represent a generic and automatic way of evaluating this type of developments, what we believe is a simple but significant contribution to the field.

The pairwise and multiple genome comparison application developed in this work (i.e. GECKO), not only represents a good heterogeneous set of tasks to

validate the system. Besides that, the two developed versions of GECKO (i.e. sequential and parallel) have significantly contributed to the bioinformatics application domain in several ways. First, GECKO has removed the limitation on input sequence length faced by equivalent software given its efficient use of the secondary storage. Second, GECKO runs faster already in its sequential version and much faster in the parallel version compared to existing methods, especially for long sequences comparison. Third, the results of GECKO has been demonstrated to be of comparable or higher quality than those reported by equivalent software. Finally, comparative studies of long sequences, which were impossible to be performed before GECKO, could yield new information in the field.

With regards to the experimental evaluation, the cloud-based HPC infrastructure has been setup in order to optimise a number of different performance metrics. The good performance has been discussed in Chapter 5, where also the limitations of the current configuration were introduced. Obviously, all the metrics can not be optimised at the same time, after all it is a compromise between the different metrics. As already mentioned, the presented system has been configured to optimise the average jobs queued time, makespan, throughput and resources utilisation respectively. However, a simple re-configuration of the tasks priority factors would enable optimising the metrics in a different order or even optimise another ones.

## Future work

We have proposed a cloud-based HPC infrastructure presenting solutions for typical tasks such as user authentication, data management, tasks scheduling and resources auto-scaling, and user-friendly exploitation of resources. However, there is still work to be done in relation to the use of cloud resources as HPC infrastructures. We can point out the following future work paths that we think are worth exploring in the future:

- Currently, the decision of when to launch a new VM to be included in the TORQUE cluster is somewhat simple. It just takes into account if there are idle jobs in the TORQUE queue and the walltime of the jobs. Ignoring other aspects of the job such as the required amount of resources would lead to provision new VMs later than required. This is to say, that if there are for instance 3 nodes and 1 job requesting 4 nodes in the queue but having a short walltime (less than the VM creation threshold), then the system should not wait to add a new VM because it already knows that it will be required. Future work could be focused on improving the previously



mentioned decision mechanism based on the amount of requested resources and its historic values by implementing machine learning algorithms using such data.

- Similarly, the VM deletion decision has been simplified. A dynamic VM is scheduled to be freed up after a configurable *idle* time period, even considering the invoicing periodicity of the cloud vendor. However, a bad decision in this point could seriously influence the system behaviour under scenarios with high variability of the workload such as periodic workloads. In such a scenario, if the VM deletion threshold is shorter than the workload periodicity, the auto-scaling technique will end unnecessarily deleting and creating new VMs because of the short periods without tasks. Studies on the probability of having more tasks in a short period of time would allow enhancing the performance of the auto-scaling technique.
- The chosen distributed resources manager has been TORQUE alongside the Maui job scheduler. System administrators might like to use other resources manager because of having a higher affinity with other solutions or if it appears a new one with remarkable performance results. The system architecture has been designed in such a way that these tools can be replaced requiring in the first place the configuration of VM instances with the new tools installed, and in second place, new Heat deployment templates to automatically instantiate the execution nodes. In addition, the auto-scaling technique should be modified to understand the output of the commands reporting the nodes statuses and the list of jobs waiting in the queue to be executed.
- The configuration parameters of the scheduler have been selected in order to improve the system behaviour within the devised cloud computing infrastructure and for the synthetic and real-world workflows presented in Chapter 5. There is not a clear limitation in this point, the set of tasks used to test the system has been big and heterogeneous enough to illustrate the advantages and limitations of the system. However, further tests with more real-world workflows would reinforce its performance. In any case, with a better or worse performance, the system can actually execute any kind of application.
- A last future work path could be the deployment and use of the cloud-based HPC infrastructure in different cloud environments such as the OpenNebula private cloud infrastructure. Although the system is theoretically prepared to be used with any other IaaS cloud solution, it has not been tested in such

infrastructures. In principle, the developed cloud management API is compatible with the OpenStack and Amazon Web Services APIs. For incompatible APIs, a simple adaptation of the cloud management API component would be required.

We would like to point out that all the research objectives planned at the beginning have been achieved. These objectives were:

- Tasks scheduling strategy with dynamic load distribution and auto-scaling of the computing resources to meet the demand.
- Strategy for data storage, transfer, and more generally, data management in the cloud environment.
- Set of complex and heterogeneous applications with data dependencies to demonstrate the effectiveness of the devised system.
- Interconnection of the cloud-based computing infrastructure with user-friendly software clients for the easy exploitation of the cloud resources.

From the academical point of view also all the set objectives at the beginning have been accomplished. These academic objectives were:

- Work experience with cloud computing environments.
- Experience with Big Data applications using HPC computing strategies.
- Acquiring know-how in tasks scheduling strategies.
- Experience in the comparative genomics and biomedicine application domains.
- Work experience in foreign teams, acquired in the scheduled stay.

After this thesis a real, powerful, efficient, configurable and easy-to-use cloud-based HPC environment is available. It is worth mentioning that the implemented strategies have been validated and are being used in the framework of the European Union Seventh Framework Programme (FP7) project Mr.Symbiomath (grant agreement number 324554), and are currently being considered in the initial implementation stages of the pan-European ELIXIR-EXCELERATE project (INFRADEV-1-H2020 Code 676559).

# Appendix A

## Cloud computing features

---

In this appendix the most distinguishing features of the cloud computing environment are described. Next sections describe the on-demand, pay-per-use, elasticity, and maintenance and upgrading outsourcing properties.

### A.1. On-demand

One of the basic features defining cloud computing is the delivery of computing and storage resources whenever the users need them. This capability removes the need of planning ahead, buying and installing the resources estimated to be used in the future. This reduces the cost not only of buying the hardware infrastructure, but also the derived costs of hosting and maintaining unused resources.

Apart from the consequences incurred in the financial point of view, this feature provides further advantages. For example, it allows software vendors to develop their software without worrying beforehand about the specific number of customers their application would have. Even in the case of having more users than initially planned, the underlying application can be up-scaled on-demand to fit the growth in number of users. This feature, as previously stated, avoids having underutilised resources with low computational load.

### A.2. Pay-per-use

An additional new aspect of the cloud environment is its billing model. From the point of the customers, they pay for the amount of time they are actually using the resources. This translates the CAPEX of buying the required hardware

infrastructure, into operational expenses OPEX measured as the time cloud resources have been used.

The way cloud computing is offering resources is related to the idea of utility computing. The similarity resides in the fact that both ideas are providing computing resources on-demand, equivalently to how a utility company provides electricity, gas or water. The main difference with regular utility companies is that water, gas or electricity can be somehow stored and later provided to the customers. However, the unused computing cycles of the cloud environment cannot be used in the future, therefore it is very important for cloud providers to efficiently use their resources.

Cloud providers such as Amazon, Microsoft or IBM, have different granularity in the way the use of resources are metered. In most of the cases, the use of resources is measured in complete-hours, but currently this fact is changing, with some providers such as Microsoft charging at the level of minutes <sup>1</sup>. In addition to the “used time”, the total amount of money to be paid depends also on a fixed price associated to the type of machine the user has asked for. Only Amazon provides resources with dynamically changing prices (i.e. spot instances <sup>2</sup>). Their price will depend first on a bid (or maximum price the user is willing to pay), and second on the available unused instances in the market. Regardless of the prices, the billing cycle usually does not change (a monthly basis is the most common).

### A.3. Rapid elasticity

Cloud providers offer different type of resources to meet user requirements. The quality of the offered service is ensured by a SLA, which is signed by providers and customers at instantiating time. The SLA is important from the point of view of the users, but it is equally important for the providers. Providers have no infinite resources, therefore with this agreement they have a base limit with the minimum quality they should be offering. With this base limit they can make a rough estimation of the maximum users they can host per each of the physical machines they have. Despite having several type of resources to meet user requirements, these requirements frequently vary along time. Cloud providers offer automatic techniques to either up-/down-scale the resources depending on the workload.

---

<sup>1</sup><https://azure.microsoft.com/en-gb/pricing/>

<sup>2</sup><http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-spot-instances.html>

## A.4. Maintenance and upgrading

Cloud providers rather than customers maintain the underlying physical computing resources. This turns into an effective outsourcing of maintenance tasks (either hardware or software). In case of hardware failures or scheduled maintenance tasks, virtual machines are migrated to different computing resources, thus not affecting the customers' experience. In this case or in similar situations, users may notice short pauses in the service but in any case they would have long service disruptions. These maintenance tasks are also contained in the SLAs to ensure they are not frequent.



# Appendix B

## Scheduling

---

This appendix contains a basic background on tasks scheduling. It first introduces the traditional types of processes to which the different tasks of a given system belongs. Typically, the scheduling decisions are based on the analysis of the type of processes a system handles. Secondly, the classical three-level categorisation of scheduling algorithms in short-term, medium-term and long-term schedulers is presented. Next, an additional categorisation of the different scheduling strategies according to their specific properties is included. Finally, 3 traditional scheduling algorithms are briefly described.

### B.1. Types of processes

Traditionally, computing processes have been categorised as either CPU-bound or I/O-bound. CPU-bound processes spend most of their execution time doing calculations, and rarely perform I/O operations. In contrast, most of the execution time of I/O-bounded processes is spent in I/O operations, with infrequent CPU bursts. Normally, schedulers aim to balance the two types of processes in order to have a similar number of processes in the ready queue and in the I/O waiting queue. Algorithms producing an unbalanced set of tasks would therefore report a bad performance. Consequently, scheduling algorithms available in the state-of-the-art have been traditionally designed to have the best possible balanced set of tasks.

Similarly, an additional categorisation separates computational processes in regular and irregular. The parallelization and scheduling of regular applications is somewhat easy, given their simple behaviour and significant degree of latent parallelism. However, the situation is more difficult for irregular applications,

which exhibit highly variable execution performance due to unpredictable memory access patterns and/or network transfers, and data imbalances.

The mentioned process categories have been usually the ones influencing the scheduler decisions. Nevertheless, presently there are more types of processes derived from the proliferation of new devices such as Graphical Processing Unit (GPU)s and Field Programmable Gate Array (FPGA)s. Such devices were originally designed for different purposes, but currently they are being used as complementary execution devices to speedup the computation. Therefore, scheduling decisions of new algorithms depend also on the devices the processes are targeting.

## B.2. Scheduling level

### B.2.1. Short-term

The short-term scheduler decides which of the processes waiting in the ready queue is executed. A scheduler iteration is executed every time the CPU receives a signal (e.g. clock and I/O interruptions or system calls). Thus the periodicity of the short-term scheduler is the lowest one. In case of no interruptions, it makes a scheduling decision every time slice, which is typically very short. Typically, is a preemptive scheduler, implying that it may remove processes from the CPU replacing them with another process. The reasons of having a preemptive scheme is first to enhance users' experience because their jobs will progress faster, and second to have a fair system where all the processes have a chance to get a CPU slice.

### B.2.2. Medium-term

The medium-term scheduler is in charge of moving processes from main memory to secondary storage (such as the hard drive). This scheduler has been traditionally very important because most systems had a little amount of main memory. However, nowadays with the increase in the available main memory it is becoming less important. Usually, the reasons why processes are moved to secondary storage vary. For example, it can be because the process is consuming a significant part of the main memory, therefore is moved out to free up memory for other processes. Another reason could be that the process has not been active for some time and new higher-priority processes have appeared in the system.

In addition to the systems with high amount of main memory, in modern op-



erating systems the role of the medium-term scheduler has significantly changed. Current operating systems manage the physical memory by using the concept of virtual memory. This memory management technique maps the virtual addresses used by a program to the physical addresses in the physical memory. The virtual addresses are organised in pages, which are swapped-in and swapped-out as the system evolves. Consequently, in the current scenario, the medium-term scheduler is not just swapping processes, it swaps pages possibly containing more than one process.

### B.2.3. Long-term

The long-term scheduler, also known as admission scheduler, manages the admission of processes to the ready queue. In other words, it is the scheduler deciding when a process execution should take place. This scheduler will therefore determine the degree of concurrency of a given system. In addition, it will decide the balance between the different types of processes (see Section B.1).

Long-term scheduling is especially important in clusters, supercomputers and HPC infrastructures in general. In such systems, it is of great importance to control the processes entering the system for execution, considering that they are expensive infrastructures that should be efficiently used. Figure B.1 illustrates the parts of a computing system where each of the scheduler types act.

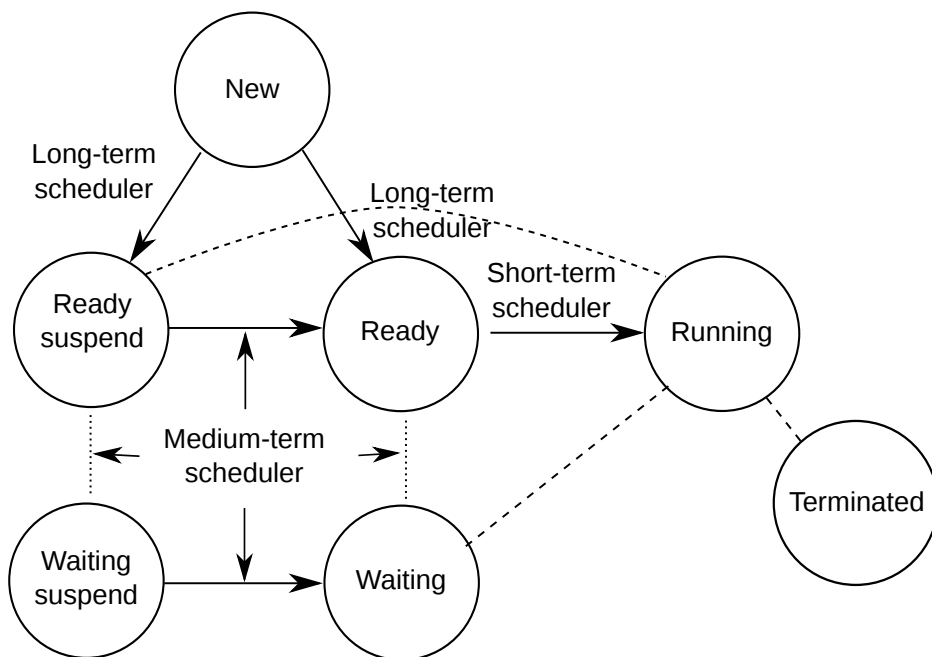


Figure B.1: Illustration of in what part of a computing system a short-term, medium-term or long-term scheduler acts.

### B.3. Types of scheduling algorithms (static and dynamic)

In the literature there exist lots of scheduling algorithms with different features each. In order to better classify the set of available algorithms, Casavant & Kuhl [17] introduced a taxonomy with the common features the methods are sharing. The taxonomy (shown in Figure B.2) has been kept as simple as possible yet containing a good set of features to classify them. Next paragraphs describe the main features, located in the backbone of the classification tree.

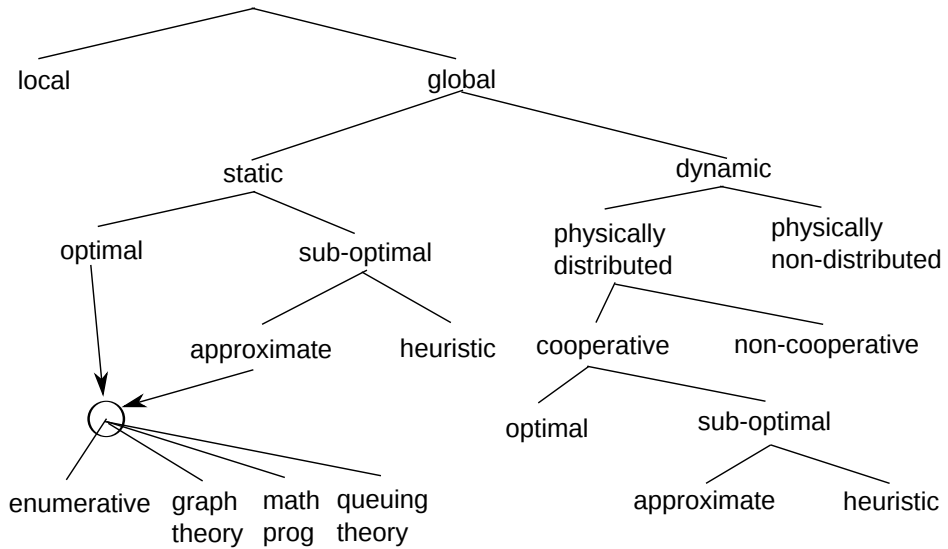


Figure B.2: Taxonomical classification of task scheduling algorithms [17].

- a) Local versus global: Local scheduling comprises the algorithms assigning processes to time-slices in single-processor systems. Instead, global scheduling includes schedulers deciding where to execute a process. To alleviate the work of global schedulers, the task of assigning the processes to processors' time-slices is left for the local scheduler of the operating system. This does not mean that global schedulers are formed by a single centralised authority, indeed they might be composed of several parts.
- b) Static versus dynamic: Static and dynamic scheduling refers to the time at which the scheduling decision is made. Static schedulers assume that the information about the processes to be run in the system is known beforehand. Considering this, they create a static plan assigning tasks to particular processors. In the case of dynamic scheduling, the algorithms consider that not all the information is known *a priori*, therefore this type of schedulers make dynamic decisions as the system evolves.
- c) Optimal versus sub-optimal: In the hypothetical case that schedulers are aware of the whole system status and the needed resources by all the process, an optimal scheduling can be made attending to one or several optimisation criteria. However, typically the computational complexity of such algorithms is very high, thus it is usual to provide sub-optimal solutions following a best-effort approach.

- d) Approximate versus heuristic: Approximate schedulers aim to reduce the scheduling computational space by exploring it until they find an acceptable solution. Whether a solution is taken as acceptable or not depends on a series of factors such as the availability of an objective function to evaluate the solution, and the time required to evaluate a solution. The heuristic category comprises algorithms making use of special parameters indirectly affecting the system performance. For example, grouping together processes which heavily communicate with each other and physically separating processes with heavy parallelism requirements.
- e) Distributed versus non-distributed: Another important aspect differentiating scheduler algorithms is the fact if the scheduling decisions are made by a single processor or if it involves several physically distributed nodes. From an abstract point of view the difference resides on where the scheduling decision is made.
- f) Cooperative versus non-cooperative: Within the distributed scheduling algorithms we can further categorise them into cooperative and non-cooperative schedulers. The cooperative algorithms involve cooperation between the different distributed components. Instead, the non-cooperative algorithms perform the decision in an independent component. It is important to notice that in cooperative algorithms the decision each distributed component is making, affects the whole system. Therefore, it is important that these components think about their local performance, but also paying attention to the global behaviour.

## B.4. Traditional scheduling algorithms

### B.4.1. FIFO

The First-In First-Out (FIFO) scheduler, also called FCFS, is the simplest but yet effective scheduling algorithm. The algorithm simply stores the processes in a queue as they arrive, dispatching first the oldest process. Its main drawback is that the throughput, turnaround and waiting times might be high because long processes can hold the CPU. In addition, there is no prioritisation so there could be problems meeting the deadline of the processes. The advantage of the lack of prioritisation is that as long as all the processes complete their execution there is no starvation.

### B.4.2. Shortest Job First

Another traditional scheduler is the Shortest Job First algorithm. It has a queue to store the processes sorted/prioritised by their execution time. At each scheduling cycle, the algorithm dispatches the process with the shortest execution time. This scheduler faces several problems. Perhaps the most notorious disadvantage from the point of view of the user is the process starvation, since is undetermined how many shorter and therefore high-priority processes may enter the system. This is particularly problematic in the case of a preemptive scheme, because running processes might become continuously interrupted by the arrival of short processes. Another important problem faced by this scheduler is that the waiting time strongly varies depending on the execution time of the processes. The positive side of this scheduler is that the throughput is usually high in most scenarios.

### B.4.3. Round-robin scheduling

This type of schedulers assigns a fixed CPU time slice to each process. The scheduler iterates over the ready queue moving out the process when its time is consumed and entering the next process in the queue. The result is good average response and waiting times, dependent on the number of processes and not on the process length. An additional advantage in this type of schedulers is that starvation never occurs, since there is no priority, thus every process has its time slice sooner or later. The main disadvantage of the round-robin scheduler is its overhead, particularly for small time slices since it causes a lot of processes interchanges.



# Apéndice C

## Resumen en español

---

### C.1. Introducción

Los numerosos avances tecnológicos en la adquisición de datos, permiten la producción masiva de grandes cantidades de datos en diversos campos que van desde la astronomía, la agronomía, la salud, hasta las redes sociales. A pesar que estos datos ofrecen muchas posibilidades a los grupos de investigación y a las empresas para efectuar estudios que permitan ampliar el conocimiento en sus respectivos campos, solo una pequeña parte de ellos –la punta del proverbial iceberg– es sintetizada, gestionada y procesada, proporcionando un conocimiento parcial del proceso que se observa. La falta de capacidad de procesamiento es el mayor cuello de botella en la obtención de resultados.

Una alternativa interesante y cada vez más importante por su disponibilidad, es el uso de estrategias de Computación de Alto Rendimiento (CAR) para afrontar el ritmo actual de generación de datos, en términos de la gestión de los mismos, la forma de acceder a ellos, así como de la planificación y distribución de las tareas que los procesan y analizan. En general, las técnicas de CAR requieren del uso de grandes y costosas infraestructuras de cómputo y almacenamiento, lo que no suele estar al alcance de muchos de los grupos que requieren de estos recursos.

La computación en la nube (*Cloud Computing*) aparece como una alternativa sugerente en la que los usuarios pagan por el uso de los recursos que consumen y cuando los necesitan. Aparte del ahorro en infraestructura, la computación en la nube ofrece otras ventajas como el ahorro en instalación, mantenimiento y suministros; la posibilidad de conseguir mejor hardware del que el usuario puede

comprar; la flexibilidad para el escalado de los recursos a usar dependiendo de las necesidades; y una mayor tolerancia a fallos, entre otras.

Dados los grandes volúmenes de datos y la carga computacional asociada a su gestión y procesamiento, la utilización eficiente de estos recursos se antoja una tarea fundamental y nada sencilla si queremos obtener el mayor rendimiento posible. Sin dejar de lado los posibles costes ocasionados por el almacenamiento y movimiento de datos, en el uso de la computación en la nube, es especialmente importante el aprovechamiento de los recursos de cómputo que se están usando, pues ello determinará, además del tiempo de respuesta para la obtención de resultados, los costes monetarios a ser sufragados. Por ello es importante el desarrollo de técnicas genéricas de planificación de tareas, que permitan un mejor aprovechamiento de los recursos computacionales.

Por otra parte, uno de los mayores inconvenientes para la adopción de estas nuevas tecnologías es la dificultad que presenta su instalación, configuración y uso. Por ello es importante facilitar estas tareas mediante el desarrollo de herramientas/aplicaciones (clientes software) que permitan el uso de las herramientas de bajo nivel desarrolladas, actuando de interfaz entre ambas partes.

La genómica comparativa, relacionada con la biomedicina, no escapa a estas consideraciones de producción masiva de datos, la alta demanda de procesamiento y sus costes asociados. Muchas de las aplicaciones en el área hacen uso de datos en el rango de GB, TB e incluso PB tanto de entrada como de salida. Por otra parte, las tareas a ejecutar en este dominio son complejas, heterogéneas, con dependencias entre ellas, usualmente formando flujos de trabajo, lo que las hace apropiadas para ser usadas como conjunto de pruebas para validar las técnicas de transferencia de datos, almacenamiento, distribución, balanceo y en general planificación de tareas en entornos de CAR que se abordan en esta tesis.

Esto hace que la investigación en técnicas que ofrezcan soluciones a los retos del procesamiento de grandes conjuntos de datos en la nube con aplicaciones en el dominio de la biomedicina y la bioinformática sean de interés directo en ciencias de la vida, medicina y en particular, salud ciudadana. Desde el punto de vista tecnológico, este trabajo aborda retos en transferencia y almacenamiento de datos en la nube, planificación de tareas, y facilitación del uso de los recursos disponibles por parte de los usuarios finales.

Se han alcanzado los objetivos que se fijaron al comienzo de este trabajo, los cuales fueron:

1. Identificación y catalogación de requerimientos para trabajar en entornos de computación en nube



2. Con los clientes, explotación más sencilla de recursos de computación en la nube por parte de los usuarios.
3. Estudio de las estrategias de planificación de tareas en entornos de computación en nube, optimización del tiempo de espera medio en cola de los trabajos, del tiempo de respuesta medio para las tareas que se vayan a ejecutar, en el uso de recursos, etc.
4. Habiendo aplicado estas técnicas al dominio de la genómica comparativa se provee a los investigadores de dicho campo de recomendaciones y herramientas que facilitan su trabajo y a su vez les permite hacer mejor uso de los recursos computacionales para acelerar la obtención de resultados, haciendo un buen provecho de las ventajas que proporcionan las infraestructuras de computación en la nube. Sin embargo, su aplicación no queda limitada a este campo como se ha podido ver reflejado con el resto de aplicaciones evaluadas.

Los resultados científicos obtenidos son los siguientes:

- Estrategia de planificación de tareas con distribución dinámica de carga de trabajo, y con escalado de los recursos de cómputo activos dependiendo de la misma.
- Estrategia para el almacenamiento, transferencia, y en general gestión de los datos en la nube.
- Conjunto de aplicaciones complejas, heterogéneas, con dependencias entre ellas (en el campo de la genómica comparativa) que demuestran la efectividad del resultado mencionado en el punto anterior, y que puede ser válido como conjunto de pruebas para soluciones equivalentes.
- Clientes software que ofrecen una interfaz de usuario amigable para la explotación de los recursos de la nube haciendo uso de los resultados obtenidos en los dos puntos anteriores.

A nivel personal, los resultados académicos que he obtenido son:

- Experiencia de trabajo con entornos de computación en la nube.
- Experiencia de trabajo con aplicaciones que trabajan con grandes conjuntos de datos y que hacen uso de herramientas de CAR.
- Experiencia en estrategias de planificación de tareas.

- Experiencia en el dominio de aplicación de la genómica comparativa y de la biomedicina.
- Experiencia de trabajo con equipos extranjeros en la estancia planificada.

Las contribuciones de este trabajo han sido publicadas en congresos internacionales con programa de revisión por pares [40, 81, 85], y en revistas científicas indexadas y en los primeros cuartiles del *ISI JCR* tal y como requieren las reglas del programa de doctorado. Adicionalmente, otras contribuciones [82, 83], aunque publicadas en congresos de más baja calidad, han ayudado a la consecución del trabajo y al entrenamiento en la escritura de documentos científicos.

Las siguientes secciones resumen los capítulos que se han ido tratando a lo largo de esta tesis empezando por el estado actual de desarrollo encontrado al comienzo de la tesis, seguido de la infraestructura basada en la nube, las estrategias de planificación de tareas y auto-escalado de recursos, evaluación experimental de las estrategias en la infraestructura descrita, y acabando con las conclusiones extraídas y posibles líneas futuras de trabajo.

## C.2. Estado actual de desarrollo

### C.2.1. Computación en la nube

Después de varios años tras la aparición del paradigma de la computación en la nube [64], todavía no hay una definición comúnmente aceptada. Sin embargo, las diferentes definiciones existentes poseen ciertas similitudes. Todas ellas están de acuerdo en que la computación en la nube comprende diferentes servicios, incluyendo almacenamiento y capacidad de cómputo proporcionados a través de la red. Asimismo, es habitualmente aceptado que la computación en la nube no es una tecnología completamente nueva, sino que se trata de una evolución de tecnologías existentes como la virtualización que ya estaba presente en la computación en *Grid*, la cual estaba pensada para resolver problemas requiriendo grandes capacidades de cómputo y almacenamiento. Una de las principales características que define la computación en la nube es su modelo de pago por uso, el cual permite traducir los gastos en infraestructura en gastos operacionales derivados del uso puntual de la misma.

Tradicionalmente la computación en la nube ha ofrecido tres modelos de servicio, los cuales proporcionan diferentes niveles de abstracción a los usuarios. Dichos modelos podrían organizarse en forma de pila, donde el nivel más ba-

jo (*IaaS*) proporciona las máximas posibilidades de configuración sacrificando la simplicidad de uso. Mientras que niveles superiores como pueden ser *PaaS* y *SaaS* abstraen en mayor medida al usuario de la complejidad inherente a la computación en la nube, a costa de perder posibilidades de configuración. A partir de los tres modelos mencionados, han ido apareciendo sucesivos modelos que proveen niveles de abstracción superiores. Ejemplos de estos modelos son las bases de datos como servicio y los flujos de trabajo como servicio.

Los entornos de computación en la nube también se pueden clasificar según los modelos de despliegue en públicos, privados, comunitarios e híbridos. Estos modelos de despliegue se distinguen por múltiples factores como pueden ser los usuarios a los que está destinado (público en general, trabajadores de una empresa o grupo de personas pertenecientes a un determinado grupo). Otro factor que los diferencia es la localización física del centro de procesamiento de datos, y si es compartido o no. Como último factor, pero no menos importante, son las necesidades de los usuarios, los cuales dependiendo de sus requisitos (normalmente relacionados con cuestiones legales) deciden si usar entornos públicos, privados, comunitarios o híbridos.

### C.2.2. Planificación de tareas

En el ámbito de la computación, el término planificación se refiere a la acción de decidir cual es el siguiente trabajo en pasar a ejecución y que recursos de entre los disponibles estará usando el mismo. La entidad encargada de realizar dicha tarea se denomina algoritmo de planificación. Cada algoritmo se encarga de optimizar uno o más objetivos, como pueden ser la reducción del tiempo medio de espera en la cola, o el aumento del volumen de trabajo por unidad de tiempo que ejecuta el sistema. Si ya es importante disponer de un buen planificador en entornos de CAR, en el campo de la computación en la nube, es aún más importante puesto que en este caso estamos pagando por cada unidad de tiempo que tenemos activo cada nodo del sistema.

Además de la planificación de tareas, en entornos de computación distribuida como *clusters*, *grids* y *clouds* se requiere de un gestor de recursos. La principal meta de dicho gestor es asegurar que los usuarios puedan usar entornos distribuidos con una dificultad similar a con la que usan máquinas locales. Aunque esta es la principal meta, los gestores de recursos distribuidos se encargan además de comprobar el estado los nodos que componen el sistema, de planificar las tareas que los usuarios envían, y de una vez planificadas asignarlas a recursos de cómputo. Un buen gestor de recursos distribuidos debe tener una serie de propiedades

como la simplicidad, portabilidad, escalabilidad, configurabilidad, expansibilidad, robusteza, seguridad y simplicidad de administración.

### C.2.3. Flujos de trabajo

La complejidad de los análisis actuales en diversos campos científicos ha hecho indispensable la interconexión de módulos más sencillos ejecutados en un determinado orden para llegar al resultado final. La ejecución orquestada de dichos módulos es lo que se conoce habitualmente como “flujos de trabajo”. El interés en los flujos de trabajo ha crecido vertiginosamente en los últimos años dado que representan una forma automática de llevar a cabo estudios complejos.

Los gestores de flujos de trabajo aparecen como una herramienta esencial para ejecutarlos de manera automática y reproducible, no sólo en ordenadores convencionales pero también en entornos de computación distribuida. Aparte de permitir la ejecución de flujos de trabajo, estos gestores también permiten su definición y el manejo tanto de los datos de entrada así como de los datos de salida (tanto intermedios como finales).

La principal razón de la popularidad de los gestores de flujos de trabajo es el número de ventajas que ofrece a los usuarios finales. En primer lugar eliminan la necesidad de ejecutar manualmente un análisis consistente en múltiples pasos. Otras ventajas de dichos gestores son la tolerancia a fallos, la grabación de la procedencia de los datos, la definición gráfica de flujos de trabajo y la exploración de resultados. Sin embargo, los gestores de flujos de trabajo poseen ciertas desventajas como la falta de portabilidad entre diferentes gestores, lo cual limita la posibilidad de usar indistintamente varios gestores al mismo tiempo; el exceso de solicitudes en los gestores públicos, que hace que la espera para llevar a cabo un determinado análisis sea significativa; y la inquietud por la privacidad y seguridad de los datos que son necesarios para el análisis, porque tienen que salir del entorno controlado del usuario final.

Además de los mencionados gestores de flujo de trabajo, tradicionalmente han existido aplicaciones con el objetivo de homogeneizar no sólo la ejecución de flujos de trabajo, sino también la de servicios disponibles en la Web provistos por diferentes áreas de investigación. Los metadatos de estos servicios (nombre, parámetros, etc.) comúnmente conocidos como *Servicios Web*, se almacenan en repositorios centrales. A partir de esta información, es posible invocarlos, aunque es necesario que los mismos tengan una interfaz de invocación común, la cual se ha diseñado e implementado en esta tesis.

### C.2.4. Dominios de aplicación

Los dominios de aplicación de las cargas de trabajo reales que han servido como prueba de concepto para la infraestructura basada en la nube diseñada son la bioinformática y la biomedicina. Más concretamente, la aplicación elegida dentro del campo de la bioinformática es la comparación de secuencias biológicas. Los algoritmos existentes para la comparación de secuencias están limitados en el tamaño de las secuencias de entrada por sus requerimientos de memoria y necesidades de cómputo. Históricamente estos algoritmos se han clasificado en algoritmos de alineamiento global y algoritmos de alineamiento local. Los algoritmos globales producen un alineamiento que tiene en cuenta todos los residuos presentes en las secuencias, mientras que los algoritmos locales determinan un conjunto de regiones similares entre las dos secuencias. De uno u otro tipo, la complejidad de dichos algoritmos junto con su mezcla de etapas intensivas en CPU y operaciones de Entrada/Salida (E/S) hacen que sean un buen caso de uso para probar mecanismos de planificación en entornos distribuidos.

Por otro lado, en el campo de la biomedicina los estudios de asociación a nivel de genoma (GWAS) son la forma en que se estudian como afectan las variaciones genómicas a las enfermedades humanas. La idea detrás de estos estudios es comparar que variaciones ocurren más frecuentemente en personas con una determinada enfermedad comparadas con las variaciones de los individuos sanos. El tamaño típico de los datos de entrada (archivo de secuenciación de genoma completo) es de 100GB, y el tamaño temporal necesario para extraer las variaciones es de 1TB. Además, si consideramos la cantidad de pacientes necesarios para obtener resultados estadísticamente significativos, los millones de variaciones que tiene cada paciente, y que existen enfermedades relacionadas no solo con una variación, sino también con más de una de ellas, estos estudios representan un problema computacional complejo con requisitos de cómputo variable, y por tanto un buen caso de uso para evaluar en entornos de computación en la nube.

### C.2.5. Trabajos relacionados

La capacidad inherente de la computación en la nube para el escalado de recursos ha hecho que tanto los proveedores públicos de computación en la nube como terceras partes hayan desarrollado estrategias de escalado automático. Las estrategias de los proveedores públicos poseen ciertas similitudes, como la agrupación de un conjunto de instancias para el cual se definen alarmas de uso de CPU y memoria que disparan mecanismos de escalado de tamaño. Estas estrategias suelen ser suficientes para la mayoría de usuarios, pero no lo son para entornos

de computación científica basados en la nube. Por ello, se han llevado a cabo diversos trabajos que usan por un lado gestores de recursos distribuidos conocidos, y por otro lado implementan estrategias de escalado automático de recursos para ajustarlos a la demanda. Por lo general, las estrategias mencionadas están limitadas a ser usadas en una plataforma de computación en la nube y un gestor de recursos distribuidos determinado.

La planificación y ejecución de flujos de trabajo en infraestructuras de computación en la nube ha seguido el camino de las estrategias que ya fueron diseñadas para entornos *Grid*, pero teniendo en cuenta las propiedades y oportunidades ofrecidas por la nube. Aunque hay diversas estrategias, existen aún ciertas cuestiones abiertas que requieren ser estudiadas. Por ejemplo, la mayoría de las técnicas trabajan con un entorno de computación en la nube simulado (cloudsim), por lo que no se trata de una infraestructura real donde poder ejecutar los flujos de trabajo *a posteriori*. Además, al tratarse de un entorno virtual se simplifica el número de características simuladas lo que puede producir diferencias comparado con un entorno real. Otro aspecto importante que no contemplan algunas estrategias es las propiedades (CPU, memoria, etc.) de la instancia requeridas por el trabajo, dando por hecho una uniformidad probablemente inexistente en las tareas a ejecutar.

## C.3. Infraestructura

### C.3.1. OpenStack

OpenStack [69] ha sido elegida como la solución de computación en la nube dentro de las disponibles y más apropiadas (modelo IaaS) para el contexto de la computación científica. Aparte de ser más apropiadas, estudios previos demuestran que dicho modelo reporta un mejor rendimiento en aplicaciones científicas. OpenStack es una solución de software libre popular que dispone de soporte de grandes empresas como IBM. En este trabajo se usa una configuración afinada de OpenStack en términos de autenticación, cómputo, gestión de datos e infraestructura de red.

### C.3.2. Autenticación

El componente de autenticación de OpenStack, llamado Keystone, se encarga de gestionar las autenticaciones de usuario a través de un Servicio Web. De

forma similar a los grupos de usuario de los sistemas operativos convencionales, OpenStack tiene el concepto de *tenants* lo que permite asignar permisos al nivel de grupo. El componente de autenticación ha sido configurado para trabajar con un árbol de credenciales de usuario LDAP [31]. El manejo de los datos representados en dicho árbol, los cuales son similares al concepto de organizaciones virtuales de los entornos *Grid*, se lleva a cabo con la aplicación DirGrid. De este modo los usuarios tienen un único par usuario-clave que recordar, mapeado con distintos permisos a los diferentes componentes del sistema. Desde el punto de vista del administrador del sistema, también se simplifica la gestión teniendo una localización centralizada para el manejo de credenciales y permisos.

### C.3.3. Gestión de datos

OpenStack separa sus sistema de almacenamiento en diferentes servicios: Glance para las imágenes de instancias, Cinder para los volúmenes/discos virtuales, y Swift para los contenedores de objetos. Glance almacena los metadatos de las imágenes, Cinder permite asociar volúmenes a instancias en ejecución, y Swift permite almacenar ficheros no necesariamente asociados a instancias en ejecución.

Al igual que para la autenticación, OpenStack es configurable con respecto al sistema de ficheros base a utilizar. En este caso se ha utilizado el sistema de ficheros distribuido Ceph [97]. Su mecanismo distribuido de resolución de localización de datos hace que no disponga de un punto único de fallo. Ceph además implementa una interfaz de programación que permite una conexión directa con los componentes de almacenamiento de OpenStack.

Uno de los objetivos de esta tesis es permitir la gestión de grandes conjuntos de datos. Para permitirlo, se ha hecho uso del protocolo GridFTP [3], el cual ha permitido usar otras tecnologías de gestión de datos como GO [29]. GridFTP fue diseñado originalmente para entornos *Grid* por lo que para ser usado en entornos de computación en la nube requiere de cierta configuración adicional. El primer punto a tener en cuenta es la autenticación, la cual se ha llevado a cabo usando las credenciales almacenadas en el árbol LDAP, y MyProxy [66] y SimpleCA para generar certificados temporales de autenticación. Una vez configurada la autenticación, el siguiente paso fue conectar el almacenamiento de objetos de OpenStack con GridFTP, puesto que este último trabaja con sistemas de ficheros convencionales. Esta conexión se ha realizado con CloudFuse, un demonio que se encarga de montar el almacenamiento de objetos del usuario autenticado en el sistema de ficheros del servidor GridFTP. El último paso necesario para configurar

el almacenamiento es el registro del servidor GridFTP configurado dentro de GO para de esta forma aprovechar las ventajas que proporciona GO (como por ejemplo la parada/reanudación de transferencias).

### C.3.4. Cómputo

Nova es el componente de cómputo principal de OpenStack. El entorno de computación en la nube usado ha sido configurando para trabajar con la tecnología de virtualización KVM<sup>1</sup>. Otro componente de cómputo de OpenStack es Glance. Glance se encarga de almacenar los metadatos de las imágenes de instancias, imágenes que son posteriormente utilizadas para añadir nuevas máquinas. En conjunto, Nova recibe la petición para añadir una nueva máquina con unas características determinadas, y Glance se encarga de recuperar la imagen o plantilla que ha pedido el usuario.

TORQUE<sup>2</sup> ha sido el gestor de recursos distribuidos elegido, el cual es capaz de gestionar grupos de máquinas con miles de nodos y trabajos, y tareas requiriendo una gran cantidad de recursos. Además de los mecanismos de planificación que trae por defecto, TORQUE permite su configuración para usar planificadores externos como Maui<sup>3</sup>, el cual ha sido usado en esta tesis. Sin embargo, TORQUE fue diseñado sin tener en cuenta las propiedades de los entornos de computación en la nube, por lo que es necesario el uso de soluciones externas para el escalado automático de los recursos de cómputo.

Otro objetivo importante fijado al comienzo de este trabajo es la simplificación del uso de este tipo de entornos por parte de usuarios con no necesariamente el conocimiento necesario de los entornos de computación en la nube. Para satisfacer este objetivo en primer lugar se diseñó una interfaz de ejecución común REST, la cual es independiente de la aplicación final. De esta manera clientes software existentes como jORCA [61] o mORCA [25] pueden hacer uso de estas aplicaciones a través de dicha interfaz abstrayendo a los usuarios de la complejidad inherente a la infraestructura.

Frecuentemente, en diversos dominios de aplicación es necesaria la interconexión de diferentes aplicaciones para llevar a cabo un análisis formando lo que se conoce como flujos de trabajo. Existen multitud de gestores de flujos de trabajo siendo Galaxy [1] uno de los más representativos al tratarse de una herramienta web que permite la definición y ejecución de flujos de trabajo de manera reprodu-

---

<sup>1</sup><http://www.linux-kvm.org/>

<sup>2</sup><http://www.adaptivecomputing.com/products/open-source/torque/>

<sup>3</sup><http://www.adaptivecomputing.com/products/open-source/maui/>



cible. Se ha configurado Galaxy para que se comuniquen con el gestor de recursos TORQUE, de tal manera que se permita la sencilla explotación de los recursos disponibles a través de interfaces gráficas. El despliegue de Galaxy ha sido automatizado, permitiendo de este modo iniciar y detener el entorno de computación en la nube en el momento que se requiera.

### C.3.5. Infraestructura de red

El componente encargado de la gestión de la infraestructura de red en OpenStack se denomina Neutron. Este componente se encarga de asignar automáticamente direcciones IP internas a las máquinas, y permite a su vez la asignación por parte del usuario de IPs públicas accesibles desde el exterior. Además, permite su configuración con diversos *plug-ins* para conectarse con los mecanismos de red de Linux o con redes definidas a nivel software.

En nuestro caso, el componente Neutron ha sido configurado para permitir controlar redes virtuales a usuarios sin privilegios administrativos autorizados por un administrador. El sub-componente que los usuarios puede administrar es la red interna a nivel de proyecto, la cual conecta las máquinas de un proyecto determinado. Las redes externas al proyecto, como puede ser Internet, solo pueden ser gestionadas por un administrador. Asimismo los usuarios con privilegios administrativos pueden añadir enrutadores virtuales para conectar diferentes redes de distintos proyectos o bien para proporcionarles acceso a redes externas.

### C.3.6. Horizon

OpenStack proporciona una consola central de gestión de los diferentes componentes (almacenamiento, cómputo, etc.). Dicha consola central llamada Horizon se trata de un portal web en el cual se pueden llevar a cabo tareas como lanzar y terminar instancias, enlazar volúmenes de almacenamiento con máquinas en ejecución, asignar IPs públicas, etc. Al mismo tiempo, esta funcionalidad que ofrece el portal comentado está disponible a través de un interfaz de programación permitiendo el acceso programático a usuarios avanzados.

### C.3.7. Interconexión entre componentes

Tradicionalmente el despliegue de aplicaciones Web ha seguido un esquema con dos servidores: un servidor web y un servidor de bases de datos. Sin embargo, en entornos de computación de alto rendimiento como el presentado en esta tesis,

es necesario hacer un despliegue más sofisticado que explote las características de la computación en la nube. Para identificar las necesidades de los distintos componentes los hemos categorizado como síncronos y asíncronos. Los servicios síncronos son aquellos para los cuales el usuario espera un tiempo de respuesta pequeño como por ejemplo el interfaz web de Galaxy. Los servicios asíncronos son aquellos para los cuales no se espera un tiempo de respuesta corto como por ejemplo la transferencia de grandes ficheros de datos. El resultado es un conjunto de máquinas interconectadas que se auto-escalan priorizando primero los servicios síncronos y después los asíncronos. La configuración automática de los componentes y su interconexión se ha definido en plantillas OpenStack Heat.

## C.4. Planificación y auto-escalado

### C.4.1. Planificación de tareas

Los planificadores por defecto incluidos en TORQUE (p.ej FIFO y *Fair-share*) no son suficientemente eficientes para grandes infraestructuras, en primer lugar por su rendimiento y en segundo lugar por las limitadas posibilidades de configuración que ofrecen. Por ello, es interesante el uso de planificadores como Maui o Moab<sup>4</sup>, los cuales son capaces de comunicarse con TORQUE. Estos dos últimos, son planificadores basados en prioridad que permiten definir la prioridad de los trabajos atendiendo a una serie de propiedades de los trabajos. En nuestro caso se han tenido en cuenta el tiempo de ejecución solicitado al enviar el trabajo, el tiempo que lleva esperando en la cola, un ratio de estos dos últimos factores y el número de recursos que pide el trabajo. Más concretamente se han priorizado trabajos cortos, seguidos de los que más tiempo llevan en la cola, y por último teniendo en cuenta los que solicitan menos recursos para de esta forma aprovechar mejor los recursos libres del sistema. En la selección de trabajos se ha activado una optimización que permite ejecutar trabajos menos prioritarios que pueden ser ejecutados en periodos de tiempo que los nodos están ociosos y no pueden ser usados por trabajos más prioritarios por sus requerimientos en cuanto a recursos se refiere.

Una vez se ha decidido que trabajo es el siguiente a ejecutar, es necesario decidir en qué nodo(s) se va a ejecutar. En este trabajo se han priorizado los nodos a tres niveles. El primer nivel elige nodos estáticos antes que dinámicos, el segundo da prioridad a los nodos más usados y el tercero prioriza los nodos con menos recursos. El objetivo es permitir la eliminación de las máquinas dinámicas

---

<sup>4</sup><http://www.adaptivecomputing.com/products/hpc-products/moab-hpc-basic-edition/>

con el mayor número de recursos lo antes posible, puesto que estas últimas suelen ser las más caras en entornos de computación en la nube.

### C.4.2. Auto-escalado

Se ha desarrollado una estrategia de escalado automático de recursos basada en Dynamic TORQUE [104]. Esta estrategia, implementada en el lenguaje Python, permite configurar una serie de parámetros como son el mínimo número fijo de instancias del *cluster*; el número máximo dinámico de máquinas que se pueden añadir; el tiempo máximo que una máquina puede estar sin ejecutar trabajos antes de ser apagada; la lista de configuraciones en cuanto a recursos se refiere para las máquinas a instanciar; y el tiempo que toma iniciar cada uno de estos tipos de instancias para decidir si compensa o no añadir una máquina nueva dependiendo de la duración del trabajo.

La estrategia de escalado pregunta a TORQUE por las tareas que están esperando ser ejecutadas y en caso de que su duración sea mayor al tiempo que toma iniciar una nueva máquina se añade. En el caso de la eliminación de máquinas, se eliminan máquinas que hayan sobrepasado un umbral de tiempo sin ejecutar trabajos, teniendo en cuenta el periodo de facturación del proveedor para aprovecharlas al máximo.

La estrategia de auto escalado se ha desplegado al mismo nodo que los servidores de TORQUE y del sistema de ficheros NFS, mientras que el resto de máquinas solamente ejecutan los clientes NFS y TORQUE para el acceso a los datos y recibir trabajos respectivamente.

## C.5. Evaluación experimental

Otro punto abordado que ha sido abordado es la evaluación experimental de la infraestructura basada en la nube diseñada usando diferentes flujos de trabajos sintéticos y reales. Los sintéticos se tratan de flujos de trabajo usados en trabajos relacionados mientras que los tres casos reales se enmarcan dentro de los campos de la bioinformática y biomedicina. Los dos primeros casos reales son aplicaciones para el cálculo de comparaciones de parejas y múltiple de secuencias de nucleótidos (GECKO [84] y GECKO en paralelo [85]), las cuales representan una contribución importante de este trabajo en el dominio de aplicación de la bioinformática. Y el tercer caso real se corresponde con una aplicación para llevar a cabo la conversión de datos en bruto de genotipado de pacientes a un formato

de representación estándar, la cual es fruto de la colaboración con expertos en el dominio de la biomedicina.

Las métricas evaluadas han sido el tiempo medio de espera de un trabajo en la cola de ejecución, el tiempo entre que el primer trabajo entra a ejecución en el sistema hasta que el último finaliza su ejecución (makespan), el número de trabajos por unidad de tiempo, y el porcentaje de utilización de recursos. La selección de flujos de trabajo que se ha hecho pretende mostrar las ventajas y limitaciones de la estrategia de planificación diseñada comparada con la planificación FIFO.

Los resultados muestran que en el caso del tiempo de espera en cola se ha reducido considerablemente en la mayoría de los casos tal y como era el objetivo. El makespan se ha reducido en algunos casos y en otros ha aumentado debido a que el planificador diseñado prioriza los trabajos más pequeños, los cuales son beneficiosos al final de la ejecución para balancear la carga de los nodos. Para las métricas de número de trabajos por unidad de tiempo, y la de utilización de recursos ocurre un escenario similar al del makespan puesto que se tratan de métricas proporcionales a la misma.

Un segundo tipo de evaluación se ha llevado a cabo para evaluar el funcionamiento de la estrategia de escalado de recursos. Esta evaluación ha sido efectuada usando tres flujos de trabajo, los dos primeros para comparación múltiple de genomas (en secuencial y en paralelo) ambos implementados en esta tesis, y el último para la extracción de variaciones genéticas en un número determinado de pacientes. En el primer flujo de trabajo el tiempo de ejecución se ha reducido desde 16,5 horas a 2,33 usando hasta 10 máquinas. En el segundo, donde el grupo de máquinas fue escalado previamente, se han obtenido valores de aceleración y eficiencia razonables, sobre todo para el conjunto de datos que genera un mayor número de tareas. En el tercero y último flujo de trabajo el tiempo se ha reducido de 4970 segundos que ya reportaba la publicación original en una infraestructura equivalente a 1756 segundos en el grupo de máquinas escalado automáticamente.

### C.5.1. Factores que afectan la planificación y el auto-escalado

Adicionalmente, se ha llevado a cabo un estudio de los principales factores que afectan el funcionamiento de las estrategias de planificación y de escalado automático de recursos diseñadas. Estos factores son la distribución de la duración de las tareas que componen los flujos de trabajo, los errores en las estimaciones de la duración de las tareas, y los retardos según el tipo de configuración de máquina en el entorno de computación en la nube de este trabajo.

La distribución de la duración de las tareas, así como en que momento son

enviadas a ejecutar, pueden determinar en primer lugar el tiempo promedio de espera en la cola, como es el caso del planificador FIFO para cargas de trabajo con tareas de larga duración al comienzo. Además, si el número de tareas no está balanceado al número disponible de nodos, el makespan y el resto de métricas dependientes de esta última pueden verse afectadas.

Las imperfecciones cometidas al estimar la duración de las tareas afectan las decisiones del planificador puesto que basa las prioridades de las tareas en ese valor entre otros. Sería de esperar que mientras peor sean estas estimaciones peores decisiones tomará el planificador. Sin embargo, sorprendentemente en algunas cargas de trabajo este no ha sido el caso. Por ejemplo cuando la carga de trabajo tiene una distribución de duración de tareas parecida, estas imperfecciones pueden ayudar a diferenciar al planificador a elegir más fácilmente las tareas más cortas al comienzo para reducir el tiempo de espera en cola. Para el caso del makespan, el tiempo extra añadido a tareas cortas hace que su tiempo pueda llegar a ser similar al de tareas de duración media y por tanto se postergue su ejecución tras ellas desequilibrando pues la carga de los nodos. En otros casos, el tiempo extra añadido a las tareas cortas fue tan grande que hizo que se convirtiesen en tareas de larga duración y por tanto tuviesen más baja prioridad que los trabajos con duración real más larga. Este hecho produce en consecuencia un mejor balanceo con estas tareas al final de la ejecución, puesto que realmente su duración es pequeña a pesar de la estimación.

Como último factor principal afectando las decisiones de planificación y escalado automático tenemos el tiempo de aprovisionamiento de las diferentes configuraciones de máquinas. Puesto que la decisión de cuando añadir una nueva máquina depende de si el tiempo de ejecución de las tareas en cola supera o no este tiempo de aprovisionamiento, se ha llevado a cabo un estudio de cuánto tiempo lleva comenzar las distintas configuraciones de máquinas. Los valores obtenidos en este estudio han permitido reducir el número de llamadas a OpenStack para el comienzo de nuevas máquinas cuando realmente la tarea es corta.

## C.6. Conclusiones y trabajo futuro

Actualmente multitud de dominios de aplicación se enfrentan al problema de analizar grandes conjuntos de datos. La CAR aparece como una alternativa sugerente para analizarlos, sin embargo las estrategias de CAR requieren del uso de infraestructuras caras que suelen ser compartidas lo que limita el ritmo de obtención de resultados. La computación en la nube se ha convertido en una posible alternativa para el eliminar este problema, sin embargo al no estar diseñada para

ello requiere de cierta investigación en el tema.

Las contribuciones de esta tesis han sido en primer lugar la selección del modelo de computación en la nube más adecuado para entornos de CAR. En segundo lugar, se han identificado las principales tareas a realizar en entornos de este tipo (como la gestión de datos, planificación de tareas, etc.). Para llevar a cabo dichas tareas, se han usado o bien software disponible para ello o se ha desarrollado software específico. En tercer lugar, se ha diseñado una estrategia de planificación basada en prioridades que optimiza una serie de criterios de rendimiento. En cuarto lugar, se ha conectado la infraestructura basada en la nube con clientes de explotación de aplicaciones con el objetivo de facilitar el trabajo a los usuarios finales. En último lugar, la evaluación experimental del sistema demuestra el buen funcionamiento del mismo.

En cuanto a autenticación de usuarios se refiere, el uso de mecanismos establecidos como el árbol de credenciales LDAP ha permitido proporcionar seguridad y privacidad en diferentes partes del sistema, todo ello facilitando las tareas del administrador al tener una localización central de gestión de credenciales y permisos. En el tema cómputo, se contribuye proporcionando a TORQUE de un mecanismo de escalado automático de recursos cuando es usado en la nube, eliminando así la necesidad de mantener un gran conjunto de máquinas a lo largo del tiempo. Este escalado automático mejora el estado del arte porque tiene en cuenta el tipo de máquina necesaria según los recursos que pide el trabajo, y además a la hora de eliminar máquinas lo hace cuando se alcanza su periodo de facturación y no inmediatamente cuando se alcanza su tiempo máximo sin ejecutar trabajos.

Es importante destacar la flexibilidad de la estrategia de escalado, a pesar de haberse probado solo con OpenStack y TORQUE, pensamos que la forma en que ha sido estructurada permitirá el uso de otras soluciones con relativa facilidad. Otro punto a favor de esta estrategia es que al no estar incluida dentro de TORQUE, en primer lugar hace que sea transparente para los usuarios finales y no añade ninguna sobrecarga a TORQUE; y en segundo lugar hace que si la estrategia de escalado deja de funcionar el *cluster* gestionado por TORQUE podría seguir ejecutando trabajos sin ningún tipo de problemas.

El planificador basado en prioridades ha sido configurado para optimizar el tiempo de espera en cola, el makespan, el número de trabajos ejecutados por unidad de tiempo y la eficiencia de uso de los recursos (en el orden mencionado). Para la elección de nodos dónde ejecutar los trabajos, siempre se prioriza los nodos estáticos sobre los dinámicos, y dentro de los dinámicos los de menos recursos para así reducir el coste que puede conllevar en entornos de computación en la

nube.

Los flujos de trabajo usados para evaluar el sistema representan un conjunto de tareas suficientemente heterogéneo. Cabe destacar, que las aplicaciones para comparación por parejas y múltiple de secuencias de nucleótidos desarrolladas en este trabajo (GECKO, GECKO en paralelo) no solo poseen patrones de ejecución similares a las cargas de trabajo simuladas usadas habitualmente para evaluar este tipo de sistemas, sino que también representan una importante contribución al campo de la bioinformática al estar permitiendo estudios en dicho campo que hasta ahora no eran posibles por las limitaciones del software existente en cuanto a tiempo de ejecución y longitud máxima de las secuencias a comparar. En general, los resultados de la evaluación experimental muestran que las métricas de rendimiento han sido optimizadas, aunque al haber dependencias entre ellas al final es un compromiso cuál optimizar con más ahínco. Sin embargo, una simple re-configuración de los parámetros para el cálculo de la prioridad de los trabajos puede permitir optimizar las métricas en un orden diferente al mencionado o incluso añadir nuevos criterios.

Resumiendo, esta tesis ha provisto de una infraestructura de CAR basada en la nube presentando soluciones para tareas usuales como la gestión de datos, la planificación de tareas y el escalado automático de recursos. Sin embargo, hemos identificado que aún hay trabajo por hacer en relación al uso de la nube como infraestructura de CAR y que podría ser interesantes de abordar en trabajos futuros. Ejemplos de estos trabajos son la mejora de las decisiones de escalado de recursos, pruebas con otros gestores de recursos distribuidos como Slurm y otros planificadores distintos a Maui, pruebas con más cargas de trabajo para reforzar la evaluación experimental, y el uso de entornos de computación en la nube diferentes a OpenStack como Amazon u OpenNebula.

Para finalizar comentar que todos los objetivos de investigación y académicos establecidos en el plan de investigación han sido cumplidos.





# Bibliography

---

- [1] Enis Afgan, Dannon Baker, Marius van den Beek, Daniel Blankenberg, Dave Bouvier, Martin Čech, John Chilton, Dave Clements, Nate Coraor, Carl Eberhard, et al. The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2016 update. *Nucleic acids research*, page gkw343, 2016. (Cited on pages 22, 51 and 138)
- [2] Ehab Nabil Alkhanak, Sai Peck Lee, and Saif Ur Rehman Khan. Cost-aware challenges for workflow scheduling approaches in cloud computing environments: Taxonomy and opportunities. *Future Generation Computer Systems*, 2015. (Cited on page 34)
- [3] William Allcock, John Bresnahan, Rajkumar Kettimuthu, Michael Link, Catalin Dumitrescu, Ioan Raicu, and Ian Foster. The Globus striped GridFTP framework and server. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 54. IEEE Computer Society, 2005. (Cited on pages 41, 61 and 137)
- [4] Bryce Allen, John Bresnahan, Lisa Childers, Ian Foster, Gopi Kandaswamy, Raj Kettimuthu, Jack Kordas, Mike Link, Stuart Martin, Karl Pickett, et al. Globus online: radical simplification of data movement via SaaS. *Preprint CI-PP-5-0611, Computation Institute, The University of Chicago*, 2011. (Cited on page 43)
- [5] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990. (Cited on page 24)
- [6] Amazon Web Services, Inc. Amazon Elastic Compute Cloud. <http://aws.amazon.com/es/documentation/ec2/>, 2015. [Online; accessed 12-November-2015]. (Cited on page 29)

- [7] Daniel P Ames, Nigel WT Quinn, Andrea E Rizzoli, et al. Intelligent Workflow Systems and Provenance-Aware Software. (Cited on page 20)
- [8] Armbrust, Michael and Fox, Armando and Griffith, Rean and Joseph, Anthony D and Katz, Randy and Konwinski, Andy and Lee, Gunho and Patterson, David and Rabkin, Ariel and Stoica, Ion and others. A view of cloud computing. *Communications of the ACM*. (Cited on page 2)
- [9] Patrick Armstrong, Ashok Agarwal, A Bishop, Andre Charbonneau, R Desmarais, K Fransham, N Hill, Ian Gable, S Gaudet, S Goliath, et al. Cloud Scheduler: a resource manager for distributed compute clouds. *arXiv preprint arXiv:1007.0050*, 2010. (Cited on page 30)
- [10] GB Berriman, JC Good, AC Laity, A Bergou, J Jacob, DS Katz, E Deelman, C Kesselman, G Singh, M-H Su, et al. Montage: a grid enabled image mosaic service for the national virtual observatory. In *Astronomical Data Analysis Software and Systems (ADASS) XIII*, volume 314, page 593, 2004. (Cited on page 76)
- [11] Michael R. Berthold, Nicolas Cebron, Fabian Dill, Thomas R. Gabriel, Tobias Kötter, Thorsten Meinl, Peter Ohl, Christoph Sieb, Kilian Thiel, and Bernd Wiswedel. KNIME: The Konstanz Information Miner. In *Studies in Classification, Data Analysis, and Knowledge Organization (GfKL 2007)*. Springer, 2007. (Cited on page 22)
- [12] Shishir Bharathi, Ann Chervenak, Ewa Deelman, Gaurang Mehta, Mei-Hui Su, and Karan Vahi. Characterization of scientific workflows. In *Workflows in Support of Large-Scale Science, 2008. WORKS 2008. Third Workshop on*, pages 1–10. IEEE, 2008. (Cited on page 100)
- [13] Daniel Blankenberg, Gregory Von Kuster, Nathaniel Coraor, Guruprasad Ananda, Ross Lazarus, Mary Mangan, Anton Nekrutenko, and James Taylor. Galaxy: A Web-Based Genome Analysis Tool for Experimentalists. *Current protocols in molecular biology*, pages 19–10, 2010. (Cited on page 100)
- [14] Jim Blythe, Sonal Jain, Ewa Deelman, Yolanda Gil, Karan Vahi, Anirban Mandal, and Ken Kennedy. Task scheduling strategies for workflow-based applications in grids. In *Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on*, volume 2, pages 759–767. IEEE, 2005. (Cited on page 14)
- [15] T. Brisco. DNS Support for Load Balancing. RFC 1794 (Informational), April 1995. (Cited on page 61)

- [16] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, 2011. (Cited on page 34)
- [17] Thomas L Casavant and Jon G Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. *Software Engineering, IEEE Transactions on*, 14(2):141–154, 1988. (Cited on pages XIII, 124 and 125)
- [18] CERN. The Large Hadron Collider. <http://home.web.cern.ch/topics/large-hadron-collider>, 2015. [Online; accessed 12-November-2015]. (Cited on page 30)
- [19] Confluence, Atlassian. Workflow Generator. <https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator>, 2015. [Online; accessed 12-November-2015]. (Cited on page 75)
- [20] Daniel Cukier. DevOps patterns to scale web applications using cloud services. In *Proceedings of the 2013 companion publication for conference on Systems, programming, & applications: software for humanity*, pages 143–152. ACM, 2013. (Cited on page 58)
- [21] Rafael Ferreira Da Silva, Weiwei Chen, Gideon Juve, Karan Vahi, and Ewa Deelman. Community resources for enabling research in distributed scientific workflows. In *e-Science (e-Science), 2014 IEEE 10th International Conference on*, volume 1, pages 177–184. IEEE, 2014. (Cited on page 75)
- [22] Ewa Deelman, Dennis Gannon, Matthew Shields, and Ian Taylor. Workflows and e-Science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528–540, 2009. (Cited on page 19)
- [23] Ewa Deelman, Carl Kesselman, Gaurang Mehta, Leila Meshkat, Laura Pearlman, Kent Blackburn, Phil Ehrens, Albert Lazzarini, Roy Williams, and Scott Koranda. Griphyn and ligo, building a virtual data grid for gravitational wave scientists. In *High Performance Distributed Computing, 2002. HPDC-11 2002. Proceedings. 11th IEEE International Symposium on*, pages 225–234. IEEE, 2002. (Cited on page 76)
- [24] Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G. Bruce Berriman, John

- Good, Anastasia Laity, Joseph C. Jacob, and Daniel S. Katz. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005. (Cited on page 100)
- [25] Sergio Díaz Del Pino, Tor Johan Mikael Karlsson, Juan Falgueras Cano, and Oswaldo Trelles. Mobile Access to On-line Analytic Bioinformatics Tools. In *International Conference on Bioinformatics and Biomedical Engineering*, pages 555–565. Springer, 2015. (Cited on pages 23 and 138)
- [26] Ricardo Graciani Diaz, Adria Casajus Ramo, Ana Carmona Agüero, Thomas Fifield, and Martin Sevier. Belle-DIRAC setup for using Amazon elastic compute cloud. *Journal of Grid Computing*, 9(1):65–79, 2011. (Cited on page 29)
- [27] Yiqiu Fang, Fei Wang, and Junwei Ge. A task scheduling algorithm based on load balancing in cloud computing. In *Web Information Systems and Mining*, pages 271–277. Springer, 2010. (Cited on pages 19, 34 and 36)
- [28] Keith Flanagan, Sirintra Nakjang, Jennifer Hallinan, Colin Harwood, Robert P Hirt, Matthew R Pocock, and Anil Wipat. Microbase 2.0: A generic framework for computationally intensive bioinformatics workflows in the cloud. *Journal of integrative bioinformatics*, 9(2):212, 2012. (Cited on page 20)
- [29] Ian Foster. Globus Online: Accelerating and democratizing science through cloud-based services. *IEEE Internet Computing*, 15(3):70, 2011. (Cited on pages 43 and 137)
- [30] Ian Foster and Carl Kesselman. The globus toolkit. *The grid: blueprint for a new computing infrastructure*, pages 259–278, 1999. (Cited on page 8)
- [31] Ian Foster and Carl Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure*. Elsevier, San Francisco, December 2003. (Cited on pages 40 and 137)
- [32] Ian Foster, Carl Kesselman, and Steve Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputing Applications*, 15(3), 2001. (Cited on page 40)
- [33] A. Freier, P. Karlton, and P. Kocher. The Secure Sockets Layer (ssl) Protocol Version 3.0. Internet RFC 6101, August 2011. (Cited on page 41)
- [34] FU, Ming and ZHANG, Wei. Open source J2EE-based Workflow Engine JBPM Design and Implementation [J]. *Computing Technology and Automation*, 4:028, 2008. (Cited on page 52)

- 
- [35] Wolfgang Gentzsch. Sun grid engine: Towards creating a compute power grid. In *Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on*, pages 35–36. IEEE, 2001. (Cited on page 31)
  - [36] Spyridon Gogouvitis, Kleopatra Konstanteli, Stefan Waldschmidt, George Kousiouris, Gregory Katsaros, Andreas Menychtas, Dimosthenis Kyriazis, and Theodora Varvarigou. Workflow management for soft real-time interactive applications in virtualized environments. *Future generation computer systems*, 28(1):193–209, 2012. (Cited on page 19)
  - [37] Robert Graves, Thomas H Jordan, Scott Callaghan, Ewa Deelman, Edward Field, Gideon Juve, Carl Kesselman, Philip Maechling, Gaurang Mehta, Kevin Milner, et al. Cybershake: A physics-based seismic hazard model for southern california. *Pure and Applied Geophysics*, 168(3-4):367–381, 2011. (Cited on page 76)
  - [38] Arpan Gupta, Paolo Faraboschi, Filippo Gioachin, Laxmikant V Kale, Richard Kaufmann, Bu-Sung Lee, Victor March, Dejan Milojicic, and Chun Hui Suen. Evaluating and Improving the Performance and Scheduling of HPC applications in Cloud. 2014. (Cited on pages 34 and 36)
  - [39] S. Hanks, T. Li, D. Farinacci, and P. Traina. *Generic Routing Encapsulation (GRE) (RFC 1701)*, October 1994. (Cited on page 55)
  - [40] Paul Heinzlreiter, James R. Perkins, Óscar Torreño Tirado, Tor Johan Mikael Karlsson, Juan Antonio Ranea, Andreas Mitterecker, Miguel Blanca, and Oswaldo Trelles. A Cloud-based GWAS Analysis Pipeline for Clinical Researchers. In Markus Helfert, Frédéric Desprez, Donald Ferguson, Frank Leymann, and Víctor Méndez Muñoz, editors, *CLOSER 2014 - Proceedings of the 4th International Conference on Cloud Computing and Services Science, Barcelona, Spain, April 3-5, 2014.*, pages 387–394. SciTePress, 2014. (Cited on pages 5, 96 and 132)
  - [41] Robert L Henderson. Job scheduling under the portable batch system. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 279–294. Springer, 1995. (Cited on page 48)
  - [42] Hiden, Hugo and Woodman, Simon and Watson, Paul and Cala, Jacek. Developing cloud applications using the e-science central platform. *Phil. Trans. R. Soc. A*, 371(1983):20120085, 2013. (Cited on page 52)

- [43] Philipp Hoenisch, Stefan Schulte, and Schahram Dustdar. Workflow scheduling and resource allocation for cloud-based execution of elastic processes. In *Service-Oriented Computing and Applications (SOCA), 2013 IEEE 6th International Conference on*, pages 1–8. IEEE, 2013. (Cited on pages 34 and 36)
- [44] Sadeka Islam, Jacky Keung, Kevin Lee, and Anna Liu. Empirical prediction models for adaptive resource provisioning in the cloud. *Future Generation Computer Systems*, 28(1):155–162, 2012. (Cited on page 106)
- [45] David Jackson, Quinn Snell, and Mark Clement. Core algorithms of the maui scheduler. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 87–102. Springer, 2001. (Cited on page 67)
- [46] Jiahui Jin, Junzhou Luo, Aibo Song, Fang Dong, and Runqun Xiong. Bar: An efficient data locality driven task scheduling algorithm for cloud computing. In *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 295–304. IEEE Computer Society, 2011. (Cited on pages 34 and 36)
- [47] Gideon Juve, Ewa Deelman, Karan Vahi, Gaurang Mehta, Bruce Berri-man, Benjamin P Berman, and Phil Maechling. Data sharing options for scientific workflows on amazon ec2. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–9. IEEE Computer Society, 2010. (Cited on page 100)
- [48] J. Karlsson and O. Trelles. MAPI: a software framework for distributed biomedical applications. *Journal of Biomedical Semantics*, 4(4), 2013. (Cited on page 22)
- [49] Johan Karlsson, Oscar Torreno, Daniel Ramet, Günter Klambauer, Miriam Cano, and Oswaldo Trelles. Enabling large-scale bioinformatics data analysis with cloud computing. In *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium On*, pages 640–645. IEEE, 2012. (Cited on page 38)
- [50] W James Kent. BLAT—the BLAST-like alignment tool. *Genome research*, 12(4):656–664, 2002. (Cited on page 24)
- [51] J.M. Korn, F.G. Kuruvilla, S.A. McCarroll, A. Wysoker, J. Nemesh, S. Cawley, E. Hubbell, J. Veitch, P.J Collins, K. Darvishi, C. Lee, M.M. Nizzari, S.B. Gabriel, S. Purcell, M.J. Daly, and D. Altshuler. Integrated

- genotype calling and association analysis of SNPs, common copy number polymorphisms and rare CNVs. *Nature Genetics*, 10(40):1253–1260, October 2008. (Cited on page 83)
- [52] Michael T Krieger, Oscar Torreno, Oswaldo Trelles, and Dieter Kranzlmüller. Building an open source cloud environment with auto-scaling resources for executing bioinformatics and biomedical workflows. *Future Generation Computer Systems*, 2016. (Cited on page 5)
- [53] Peter Lawrence. *Workflow handbook 1997*. John Wiley & Sons, Inc., 1997. (Cited on page 19)
- [54] Young Choon Lee, Hyuck Han, Albert Y Zomaya, and Mazin Yousif. Resource-efficient workflow scheduling in clouds. *Knowledge-Based Systems*, 80:153–162, 2015. (Cited on page 75)
- [55] Song Li, Yangfan Zhou, Lei Jiao, Xinya Yan, Xin Wang, and Michael R Lyu. Delay-aware cost optimization for dynamic resource provisioning in hybrid clouds. In *Web Services (ICWS), 2014 IEEE International Conference on*, pages 169–176. IEEE, 2014. (Cited on page 106)
- [56] David J Lipman and William R Pearson. Rapid and sensitive protein similarity searches. *Science*, 227(4693):1435–1441, 1985. (Cited on page 24)
- [57] Maciej Malawski, Gideon Juve, Ewa Deelman, and Jarek Nabrzyski. Cost-and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 22. IEEE Computer Society Press, 2012. (Cited on pages 35 and 36)
- [58] Maciej Malawski, Gideon Juve, Ewa Deelman, and Jarek Nabrzyski. Algorithms for cost-and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds. *Future Generation Computer Systems*, 48:1–18, 2015. (Cited on pages 19, 75, 99, 100 and 106)
- [59] Ming Mao and Marty Humphrey. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 49. ACM, 2011. (Cited on page 14)
- [60] Ming Mao and Marty Humphrey. A performance study on the vm startup time in the cloud. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 423–430. IEEE, 2012. (Cited on page 106)

- [61] Victoria Martín-Requena, Javier Ríos, Maximiliano García, Sergio Ramírez, and Oswaldo Trelles. jORCA: easily integrating bioinformatics web services. *Bioinformatics*, 26(4):553–559, 2010. (Cited on pages 22 and 138)
- [62] Vivien Marx. Biology: The big challenges of big data. *Nature*, 498(7453):255–260, 2013. (Cited on page 1)
- [63] Massachusetts Institute of Technology. StarCluster. <http://star.mit.edu/cluster/>, 2015. [Online; accessed 12-November-2015]. (Cited on page 31)
- [64] Peter Mell and Tim Grance. The NIST definition of cloud computing. 2011. (Cited on pages 8, 9 and 132)
- [65] Saul B Needleman and Christian D Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, 1970. (Cited on page 24)
- [66] Jason Novotny, Steven Tuecke, and Von Welch. An online credential repository for the grid: MyProxy. In *High Performance Distributed Computing, 2001. Proceedings. 10th IEEE International Symposium on*, pages 104–111. IEEE, 2001. (Cited on pages 44 and 137)
- [67] Aisling O’Driscoll, Jurate Daugelaite, and Roy D. Sleator. ‘big data’, hadoop and cloud computing in genomics. *Journal of Biomedical Informatics*, 46(5):774 – 781, 2013. (Cited on page 2)
- [68] William R Pearson and David J Lipman. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences*, 85(8):2444–2448, 1988. (Cited on page 24)
- [69] Ken Pepple. *Deploying openstack*. ” O’Reilly Media, Inc.”, 2011. (Cited on pages 38 and 136)
- [70] Deepak Poola, Saurabh Kumar Garg, Rajkumar Buyya, Yun Yang, and Kotagiri Ramamohanarao. Robust scheduling of scientific workflows with deadline and budget constraints in clouds. In *Advanced Information Networking and Applications (AINA), 2014 IEEE 28th International Conference on*, pages 858–865. IEEE, 2014. (Cited on pages 35 and 36)
- [71] Sergio Ramírez. *Arquitectura SOA para la integración de servicios distribuidos*. PhD thesis, University of Malaga, 2012. (Cited on page 22)



- [72] Sergio Ramírez, Antonio Muñoz-Mérida, Johan Karlsson, Maximiliano García, Antonio J Pérez-Pulido, M Gonzalo Claros, and Oswaldo Trelles. MOWServ: a web client for integration of bioinformatic resources. *Nucleic acids research*, page gkq497, 2010. (Cited on page 23)
- [73] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. *Address Allocation for Private Internets (RFC 1918)*, February 1996. (Cited on page 55)
- [74] Bhaskar Prasad Rimal, Eunmi Choi, and Ian Lumb. A taxonomy and survey of cloud computing systems. In *2009 Fifth International Joint Conference on INC, IMS and IDC*, pages 44–51. IEEE, 2009. (Cited on page 9)
- [75] Roloff, Eduardo and Diener, Matthias and Carissimi, Alexandre and Navaux, Philippe OA. High Performance Computing in the cloud: Deployment, performance and cost efficiency. In *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, pages 371–378. IEEE, 2012. (Cited on page 2)
- [76] Vipin Samar. Unified Login with Pluggable Authentication Modules (PAM). In *Proceedings of the 3rd ACM Conference on Computer and Communications Security, CCS '96*, pages 1–10, New York, NY, USA, 1996. ACM. (Cited on page 45)
- [77] Sucha Smanchat and Kanchana Viriyapant. Taxonomies of workflow scheduling problem and techniques in the cloud. *Future Generation Computer Systems*, 52:1–12, 2015. (Cited on page 34)
- [78] Temple F Smith and Michael S Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, 1981. (Cited on page 24)
- [79] Inc. SwiftStack. The OpenStack Object Storage system Deploying and managing a scalable, open-source cloud storage system with the SwiftStack Platform, 2012. (Cited on page 43)
- [80] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: The Condor experience. *Concurrency-Practice and Experience*, 17(2-4):323–356, 2005. (Cited on page 30)
- [81] Oscar Torreno, Michael T Krieger, Paul Heinzlreiter, and Oswaldo Trelles. Pairwise Genome Comparison Workflow in the Cloud Using Galaxy. *Procedia Computer Science*, 51:2864–2868, 2015. (Cited on pages 5 and 132)

- [82] Oscar Torreno and Oswaldo Trelles. Running workflows in the cloud. In *Jornadas de Paralelismo (JP)*, 2014. Sociedad de Arquitectura y Tecnología de Computadores, 2014. (Cited on pages 5 and 132)
- [83] Oscar Torreno and Oswaldo Trelles. Auto-scaling strategy for Openstack cloud resources managed by TORQUE. In *Jornadas de Paralelismo (JP)*, 2015. Sociedad de Arquitectura y Tecnología de Computadores, 2015. (Cited on pages 5 and 132)
- [84] Oscar Torreno and Oswaldo Trelles. Breaking the computational barriers of pairwise genome comparison. *BMC bioinformatics*, 16(1):250, 2015. (Cited on pages 5, 80 and 141)
- [85] Oscar Torreno and Oswaldo Trelles. Two level parallelism and I/O reduction in genome comparisons. *Cluster Computing*, pages 1–12, 2017. (Cited on pages 5, 94, 132 and 141)
- [86] Oswaldo Trelles. On the parallelisation of bioinformatics applications. *Briefings in Bioinformatics*, 2(2):181–194, 2001. (Cited on pages 7 and 23)
- [87] Oswaldo Trelles, Pjotr Prins, Marc Snir, and Ritsert C Jansen. Big data, but are we ready? *Nature Reviews Genetics*, 12(3):224–224, 2011. (Cited on page 1)
- [88] Oswaldo Trelles and Andrés Rodríguez. *Bioinformatics and Parallel Metaheuristics*, pages 517–549. John Wiley & Sons, Inc., 2005. (Cited on pages 7 and 23)
- [89] S. Tuecke, V. Welch, D. Engert, L. Pearlman, and M. Thompson. *Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile (RFC 3820)*, June 2004. (Cited on page 44)
- [90] Ruben Van den Bossche, Kurt Vanmechelen, and Jan Broeckhove. Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads. In *Cloud Computing (CLOUD)*, 2010 IEEE 3rd International Conference on, pages 228–235. IEEE, 2010. (Cited on page 14)
- [91] Luis M Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50–55, 2008. (Cited on page 8)
- [92] Amandeep Verma and Sakshi Kaushal. Cost-Time Efficient Scheduling Plan for Executing Workflows in the Cloud. *Journal of Grid Computing*, pages 1–12, 2015. (Cited on page 75)

- [93] A. Vogel, D. Griebler, C. A. F. Maron, C. Schepke, and L. G. Fernandes. Private IaaS Clouds: A Comparative Analysis of OpenNebula, CloudStack and OpenStack. In *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, pages 672–679, Feb 2016. (Cited on page 38)
- [94] Jianwu Wang, Prakashan Korambath, Ilkay Altintas, Jim Davis, and Daniel Crawl. Workflow as a service in the cloud: architecture and scheduling algorithms. *Procedia Computer Science*, 29:546–556, 2014. (Cited on pages 9, 35 and 36)
- [95] Lizhe Wang, Gregor Von Laszewski, Andrew Younge, Xi He, Marcel Kunze, Jie Tao, and Cheng Fu. Cloud computing: a perspective study. *New Generation Computing*, 28(2):137–146, 2010. (Cited on page 8)
- [96] Wendy A Warr. Scientific workflow systems: Pipeline pilot and knime. *Journal of computer-aided molecular design*, pages 1–4, 2012. (Cited on page 22)
- [97] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, and Carlos Maltzahn. Ceph: A scalable, high-performance distributed file system. In *In Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI)*, pages 307–320, 2006. (Cited on pages 42 and 137)
- [98] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, and Carlos Maltzahn. CRUSH: controlled, scalable, decentralized placement of replicated data. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing, SC '06*, New York, NY, USA, 2006. ACM. (Cited on page 42)
- [99] Tom White. *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012. (Cited on page 31)
- [100] Michael Wilde, Mihael Hategan, Justin M Wozniak, Ben Clifford, Daniel S Katz, and Ian Foster. Swift: A language for distributed parallel scripting. *Parallel Computing*, 37(9):633–652, 2011. (Cited on pages 20 and 100)
- [101] Fuhui Wu, Qingbo Wu, and Yusong Tan. Workflow scheduling in cloud: a survey. *The Journal of Supercomputing*, pages 1–46, 2015. (Cited on page 34)
- [102] Andy B Yoo, Morris A Jette, and Mark Grondona. Slurm: Simple linux utility for resource management. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 44–60. Springer, 2003. (Cited on page 49)

- [103] Jia Yu, Rajkumar Buyya, and Kotagiri Ramamohanarao. Workflow scheduling algorithms for grid computing. In *Metaheuristics for scheduling in distributed computing environments*, pages 173–214. Springer, 2008. (Cited on pages 14 and 19)
- [104] S Zhang, L Boland, P Coddington, and M Sevier. Dynamic VM Provisioning for TORQUE in a Cloud Environment. In *Journal of Physics: Conference Series*, volume 513, page 032107. IOP Publishing, 2014. (Cited on pages 32, 70, 74 and 141)